The use of Google Apps for implementing BI applications

Omar Ghadban Pou
Menthor: Izr. Prof. Dr. Vili Podgorelec
Universitat Politècnica de Catalunya, FIB
University of Maribor, FERI
July 2010

Abstract

This project aims to prove that is possible to develop Business Intelligence applications by using Google Apps.

Instead of parameterize some existing tool, we want to demonstrate that we can build a solution for a concrete corporation according to its needs.

For this, we have separated the work in two parts. The first one is based on a studio about which Google code tools and libraries can be useful for this purpose. All the alternatives have been classified, analyzed and exemplified. Also, for concluding the studio, we have explained how we should assemble them in order to obtain a BI system. As a second part, we have implemented an example tool following the steps described on the previous studio, and have also proposed some possible improvements to apply to the original solution.

After that work we have demonstrate that we can develop totally functional BI tools using Google Apps. Moreover, these ones will be characterized by its sturdiness, fastness, security and change adaptability.

All of that allows us to say that Google Apps is an excellent platform for innovate and develop BI solutions for companies totally adapted to their needs, reducing costs, and giving a new perspective over themselves.

Index

1. Introduction

Since human beings exist the taking of decisions is an inherent part of our condition. Every day, since we wake up, we must face lots of them, from the easiest ones to those who carry a high complication indeed.

Should I have a coffee for breakfast? Should I visit my aunt? Which car should I buy? Is this a good moment to invest?

But, what's the meaning of take a decision? We can define it as the choosing between the different alternatives or solutions that a problem offers.

Understanding enterprises as a group of persons, we can imagine that decision making is something common in them too. The managing of a corporation requires day by day decide in order to increase the performance and reach the set goals. But that choosing may be based in objective information, in something that let us know the context of it and the real value of each alternative we've to solve the conflict. Without that support the decision-making process will look similar to playing rock-scissors-paper game.

Once upon a time I heard that a decision has two parts. The first one is calculate the value of each option, and the second is decide despite numbers. We could make an analogy of this to a poker game: The information gives us a good hand but we, the people, must play the game.

At this point, it will be also important to differentiate between data and information. For this, we'll use the definitions that Davenport and Prusak gave for these concepts.[10]

Data are the minimum semantic unit and correspond to primary elements of information, which by themselves are irrelevant to support decision making. It can be also seen as a discrete set of values that don't say anything about things and are not indicative for action. On the other hand, information can be defined as a set of processed data with a meaning (relevance, purpose and context) and, therefore, useful for who should make decisions by reducing uncertainty.

The following example illustrates that concept:

0034660332015 is just a set of numbers, data

But if we know that:

- *0034 is the Spanish international phone code*
- *660332015 is a cell phone number*

Adding this meta-data to data, we have the information that 0034660332015 is a Spanish cell phone number.

Only from the information we can extract knowledge, and that's what we should search and use in order to take correct decisions.

Because of its power and scope, Information Technologies can help organizations in that important process. Trough them we can gather data, analyze it, transform it into information, report it and, at the end, extract the searched knowledge for take

decisions. And this one is the objective of Business Intelligence discipline: provide the necessary knowledge to manage a company successfully.

As we will see in this project, this IT branch allows corporations to go beyond than simply go on executing its operational processes. It gives a new perspective over themselves which will let control and manage the business process, and make decisions over its course based in accurate information.

There are lots of succeeded and good Business Intelligence products in the market such as SAP's *BusinessObjects*, SAS *Business Intelligence* or IBM's *Cognos*. But what we question to ourselves is: what about developing an own BI system for a corporation taking a look to its concrete needs? And also, what about doing it optimizing resources at the same time?

At this point is where we need to turn and look at Google. As its Apps service allows developers to build IT solutions, we can see it as a platform for develop BI ones. Trough App Engine tool we have access to the same building blocks that Google uses for its own applications, making it easier to build an application.

So, what we wonder now is: what about using Google Apps for BI? How can we use it in order to build a BI system?

And that's the question this project aims to answer. For this, we've structured it in the following way: first of all, we're going to see brief descriptions about BI and Google Apps in order to contextualize it. After that, we'll perform a studio about the different suite of tools and libraries offered by Google Code, useful for develop BI systems. Finally, we'll see an example about the using of some of the explained alternatives to build a concrete BI tool.

After that, for concluding the investigation, we'll expose the extracted conclusions.

2. About Business Intelligence

The best thing we can do to introduce Business Intelligence[1] is start defining it. As is an ambiguous term that wouldn't be easy, so we'll go by parts.

First we'll take a look to a global definition about it, and then, step by step, go into technical skills.

Thus, we can consider BI as the company data transformation into knowledge to earn some competitive advantage. It comprises the known of corporation's performance and the future events anticipation, in order to support corporative decisions.

So that's why we can see it as the meeting point between business, its management and information technology support. The following image shows that concept.[13][14]

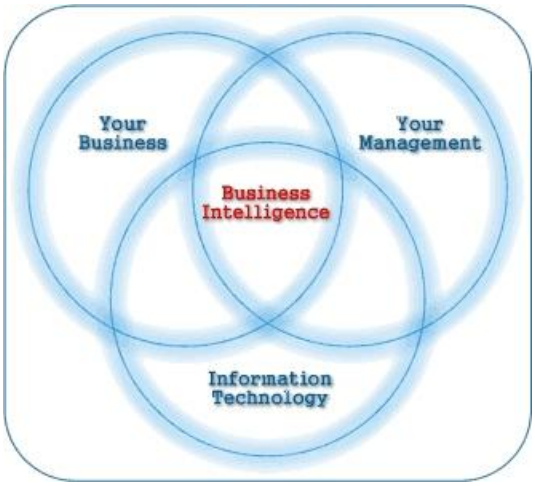


Fig.1 – BI seen as a meeting point.

On the other hand, talking in information systems terms, we can imagine organizations as 3-level systems (Fig.2) where processes are executed, and must be managed and controlled.

[1] From now until the end of the document we'll refer to it as BI.

Fig.2 – Information systems levels.

That processes generate and consume information during its execution. A part of it, known as operational information, is short-term consumed. The rest is stored in the different transactional systems (ERP, CRM, SCM, etc…) waiting to be used in the tactical or strategic decision-making.

All the data introduced into operational levels should be measured to be controlled, and controlled to be managed. BI tools deliver the information to processes managers and owners, improving its efficiency and even its efficacy.

So that's why we can also assimilate BI as a mediator to turn information into knowledge. [11][13][4]

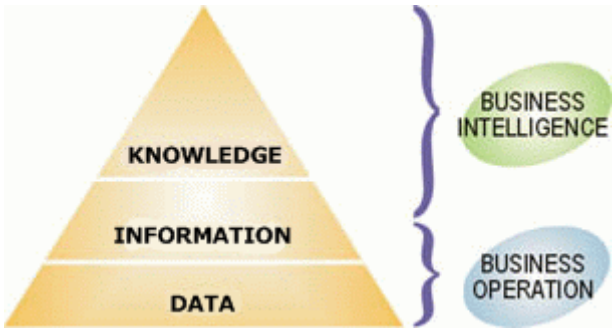


Fig.3 – Business Intelligence ambit.

Entering in technical aspects, we can see BI as the addition of disciplines, applications and technologies who offer the possibility of join, depurate and transform data from transactional systems and unstructured information into structured information for its posterior direct mining or analysis, and transformation into knowledge to support the business decision-making process.[11]

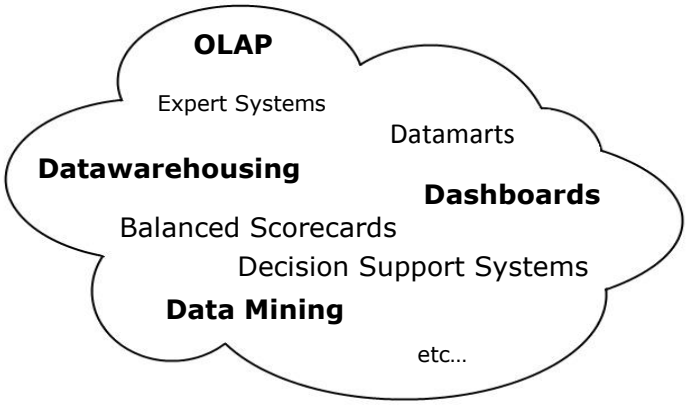


Fig.4 – Acronyms, tools and disciplines involved by BI.

Using ETL (extract, transform and load) tools and techniques, data is extracted from different sources, debugged and prepared (data homogenization) to be loaded into a datawarehouse.

In that point is ready to be used by the different BI tools in order to help the decision-making process. BI is nourished by all the company's operational systems, as well as external information, and collects, cleans, consolidates, unifies and formats the information making it ready for the consumption (Fig.5). [11][14]

All this set of tools and methodologies, have in common the following characteristics:[11]

- Control and management of business process. Ensure user access to data regardless of its origin.
- Decisional character. It’s seek to go beyond the presentation of information, so that users can have access to analysis tools that enable them to select and manipulate only interesting information, and thus, offer support in decision-making process.
- End user oriented. The aim is independence between user expertise and ability to use these tools. BI tools should “talk” business language.

According to its complexity level BI solutions can be classified as:

- Query and reporting.
- OLAP (On-Line Analytic Processing) cubes.
- Data Mining.
- Forecasting systems. Based on the studio of temporal series.

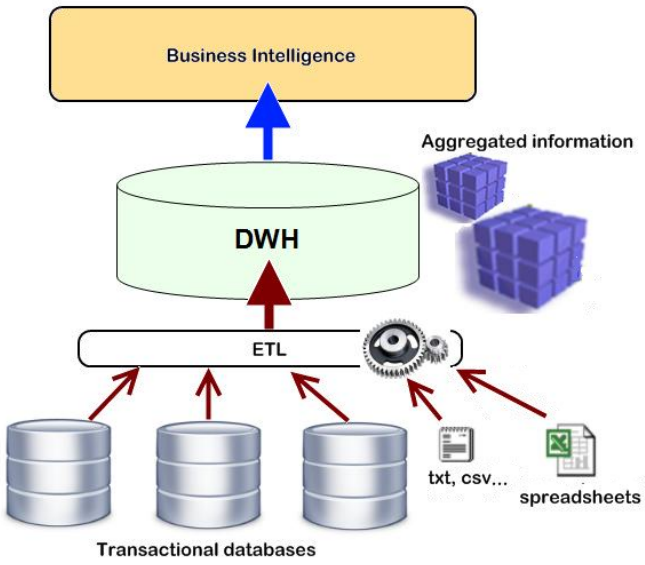


Fig.5 – Schema of the BI nourishing from its sources.

2.1. A brief History

The first appearance of the term business intelligence was in a 1958 article written by the IBM researcher Hans Peter Luhn. On it, he defined intelligence as: "*the ability to apprehend the interrelationships of presented facts in such a way as to guide action towards a desired goal*".[1]

Later, in 1989, Howard Dresner proposed BI as an umbrella term to describe "*concepts and methods to improve business decision making by using fact-based support systems".*[2] It wasn't until the late 1990s that this usage was widespread.

After seeing that brief description about the usage of the term Business Intelligence, we are going to take a look to a chronological evolution about this kind of systems.[11]

- 60's: *batch* reports

  Information is difficult to find and analyze, not flexible, each request needs to be re-programmed.

- 70's: Firsts DSS (*Decision Support Systems*) and EIS (*Executive Information Systems*)

  Terminal-based, not integrated with the rest of tools.

- 80's: Data access and integrated analysis tools (known as *intelligent business tools*)

  Query&Reporting tools, spreadsheets, integrated graphical interfaces, easy to use.

Problem: Access to operational databases (possible appearence of *killer queries* against the ERP).

- 90's: Datawarehouses and OLAP tools
- 00's: Data mining and simulation tools

Once we've seen a brief history about BI let's going to take a look to its future.

2.2. The future of BI

A 2009 *Gartner* paper predicted the following developments in the Business Intelligence market:[3]

- Because of lack of information, processes, and tools, through 2012, more than 35 percent of the top 5,000 global companies will regularly fail to make insightful decisions about significant changes in their business and markets.
- By 2012, business units will control at least 40 percent of the total budget for business intelligence.
- By 2010, 20 per cent of organizations will have an industry-specific analytic application delivered via software as a service as a standard component of their business intelligence portfolio.
- By 2012, one-third of analytic applications applied to business processes will be delivered through coarse-grained application mashups.

3. About Google Apps

In this chapter of the present document, we'll take a view over Google Apps with the objective of continue contextualizing the project. First of all, we'll talk about the company, and then explain what exactly is Google Apps.

Thus, *Google Inc.* is a multinational public cloud computing and Internet search technologies corporation established in 1998 by Larry Page and Sergey Brin. It hosts and develops lots of Internet-based services and products. For this, runs over one million servers in data centers around the world, and processes over one billion search requests and lots of user-generated data every day.

Google's rapid growth since its incorporation has triggered a chain of products beyond the company's core search engine. It offers online productivity software, social networking and developer tools. Also, its products extend to the desktop as well.[5][6]

Nowadays, is one of the biggest and most known companies all over the world. Because of its popularity and numerous products, *Alexa*[2] lists Google as the Internet's most visited website.

After this brief overview about Google, we will talk about the product in which will be focused that project: Google Apps. Thus, we can define it as a service for using custom domain names with several Google products. It features some web applications with similar functionality to traditional office suites, including: Gmail, Google Calendar, Talk, Docs and Sites. So, it provides powerful collaboration and communication browser tools for enterprises.[8]

Also, as is a cloud computing-based service, the using of it implies a reduction of IT costs, saving money in physical infrastructure.

After that, we can see Google Apps as just a suite of business productivity tools for corporations, but actually, is more than that. Trough it is possible to develop and run applications under Google's infrastructure, so it can be also seen as a platform for innovation. And that's the feature that this project aims to exploit.

All applications developed with its framework are easy to create, maintain and update when traffic and data storage needs increase. Moreover, these solutions can be also used in combination with some other Google Code APIs in order to be improved. All of it maintaining the principles of change adaptability, code reuse and loose coupling.[9]

Furthermore, the fact of working with Google Apps implies other kinds of advantages to corporations, such as a guaranteed reliability of 99'9%, big storage capabilities, accomplishment of regulations and information security, and customer assistance 24/7.[12]

[2] Alexa Internet is an Amazon.com group company (www.alexa.com). Since 1996, offers a gauge named Traffic Rank based on the number of visitors that a web site has. More hits means better positioning in that gauge.

All these features, the productivity and innovation ones, make Google Apps a platform used by so many kinds of corporations. Some examples about it are shown in the following table:

	Ambit
Group Serhs	Tourism
Salesforce.com	Software provider
RedOctane	Videogames developers
Capgemini	IT consulting
District of Columbia	Public administration
Essilor	Sanity

Tab.1 – Examples about companies using Google Apps.[7]

4. Tools and Libraries

After seeing brief introductions about the context of the project we've arrived to the main part of it. Thus, this chapter of the document is dedicated to introduce and analyze the different Google's suite of tools and libraries which can be used by developers for building BI solutions.

As Google Apps is the offered service to run and build applications this will be the starting point, but we will go beyond than that. As is said in the previous chapter, applications developed with it can use some other Google Code APIs in order to be improved. So, in this fourth chapter we will see a complete guide about tools that in combination with Google Apps can help us obtaining a complete BI tool.

We should note that it will be only focused on those tools from Google, excluding third-party compatible ones.

As a result we will have a studio about which kind of tools offered by Google can be used in order to build a BI system for some enterprise.

For this purpose, we will classify first some of the alternatives to use in combination with Google Apps in terms of their typology, either visualization either concerning the data layer. Each one will be described and also shown how to use it. Not concerning displaying or data layer ones will be classified as *Others* in the section 4.3.

After that, we will introduce the Google Apps's App Engine tool, which will be useful for build and run our application.

Finally, for concluding this chapter, we will see an overview to the structure of a BI solution developed using Google Apps.

4.1. Visualization Layer

In this first section, the two given options to display stored data in our BI system will be presented. Both belong to Google Chart Tools.

Through them, developers are enable for adding live charts to any website easily and from a variety of data sources.

Next, we are going to see the chart visualization types provided by the tool: as static images or as interactive ones. Moreover, for each, we will take a look to the different offered views to display data.

4.1.1. Image charts (aka Google Chart API)

That API allows to dynamically generate charts with a URI string. It works returning a PNG image of the desired chart in response to a URI GET or POST request.
Through it is possible to generate many kinds of charts, from pie or line ones to QR codes and formulas.

All the information about the chart we desire, such as data, size, colors and labels, are part of the URI.
Thus, all of them start with "*http://chart.apis.google.com/chart?*" followed by the parameters that specify chart data and appearance. Each chart's documentation lists the available optional parameters.

Once we get the chart URI string we can easily embed it in the code of our applications visualization part with an HTML image tag <img> pointing to it[3]. The following example shows the whole process:

http://chart.apis.google.com/chart?chs=250x100&chd=t:60,30,10&cht=p3&chl=Yes|No|Unknown

where:

Chs	*Defines de dimensions of the chart.*
Chd	*Contains the data values. For the example 60,30,10.*
Cht	*Selects the chart type, in that case a 3D pie one.*
Chl	*Contains the data tags. In the example correspond to Yes, No, Unknown.*

…
<img
src="http://chart.apis.google.com/chart?chs=250x100&chd=t:60,30,10&cht=p3&chl=Yes|No|Unknown" />
…

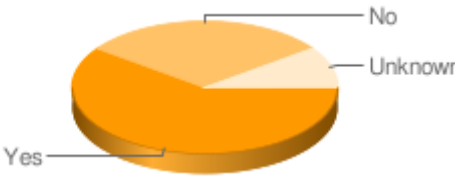


Fig.6 – Resulting pie chart.

That described process corresponds to GET method. However, URIs are limited to 2K in length on it, so if we need more data than that we can use POST instead of GET.

In this other method we also get a PNG chart as a response, but we can work with 16K of data.

[3] It will be necessary to adapt the chart URI to HTML style, substituting '&' symbols for '&'.

All the ways for using POST require additional coding, either in the page code or on the server hosting the page. It consists typically on create a separate page for each chart, and embed or link these pages in the main one using an <iframe> or <img> tag.
The image will appear as a response of an HTML form with hidden camps acting as parameters for our desired chart.

Moreover, it's also possible to use POST in combination with some other languages, such as JavaScript or php. Here is an example about using it with Javascript:

http://chart.apis.google.com/chart?chs=300x200&cht=lc&chtt=Post%20chart&chxt=x&chd=t:40,20,50,20,100

where:

Chs	*Defines de dimensions of the chart.*
Cht	*Selects the chart type, in that case a line one.*
Chtt	*Contains the chart title. For our example 'Post chart'.*
chxt	*Selects the chart axe to work on.*
Chd	*Contains the data values for the specified chart. In the example 40,20,50,20,100.*

post_chart.html

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <script type='application/javascript'>
    // Send the POST when the page is loaded,
    // which will replace this whole page with the retrieved chart.
    function loadGraph() {
      var frm = document.getElementById('post_form');
      if (frm) {
       frm.submit();
      }
    }
  </script>
  </head>
  <body onload="loadGraph()">
  <form action='http://chart.apis.google.com/chart' method='POST' id='post_form'>
    <pre>&lt;form action='http://chart.apis.google.com/chart' method='POST' id='post_form'&gt;
      &lt;input type=&quot;hidden&quot; name=&quot;cht&quot; value=&quot;lc&quot;  /&gt;
      &lt;input type=&quot;hidden&quot; name=&quot;chtt&quot; value=&quot;Post chart&quot;
 /&gt;
      &lt;input type='hidden' name='chs' value='300x200' /&gt;
      &lt;input type=&quot;hidden&quot; name=&quot;chxt&quot; value=&quot;x&quot; /&gt;
      &lt;input type='hidden' name='chd' value='t:40,20,50,20,100'/&gt;
      &lt;input type=&quot;submit&quot;  /&gt;
    &lt;/form&gt;
  </body>
</html>
```

main_page.html

```
…
<iframe src="post_chart.html" width="300" height="200"></iframe>
…
```

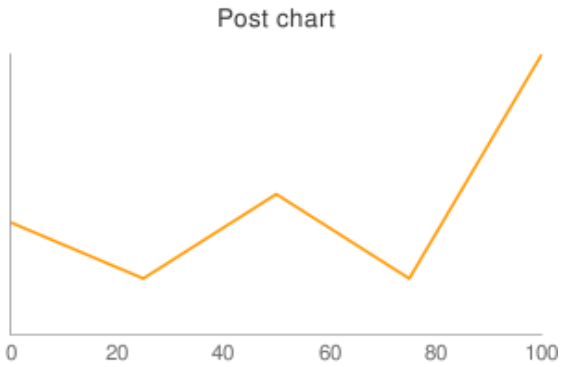
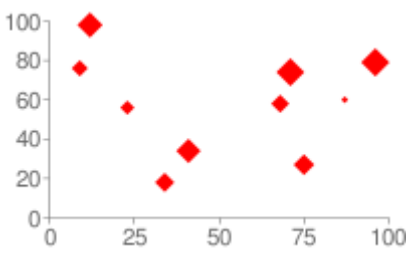
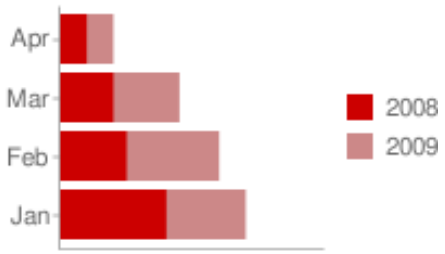
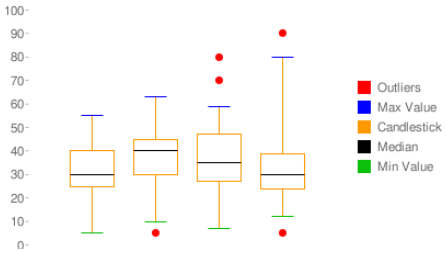
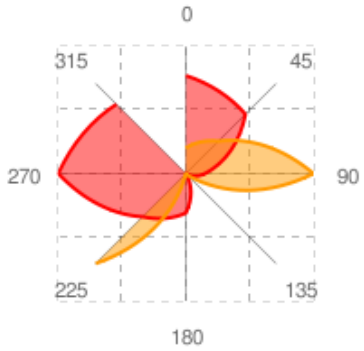


Fig.7 – Returning line chart.

Taking a look to that given example, we can see that the chart page is the one who gets the returned image and the main one just needs to load it. So even the process is not as easier as with GET method it stills being easy.

Furthermore, Google Chart API also allows us to test and parameterize the visualizations for a given chart URI before using them, in order to decide if it will be or not satisfactory for its purpose. That can be done through Live Chart Playground tool.

Next, and before concluding this section, we will take a look to some of the charts to display data offered by Chart API (Tab.2).

Chart type	
Scatter	
Map	
Bar	
Box	
Google-O-Meter	
Radar	

Tab.2 – Examples of data displayed as charts using Chart API.

4.1.2. Interactive charts (aka Google Visualization API)

The second of the alternatives given by Google Chart Tool in order to display data is the Visualization API. Unlike in the previous section, charts developed with this API can easily access to stored data, don't have data size limitation and support also user interaction.

Trough it, we are allowed to expose data located on any data-store that is connected to the web, as a Visualization compliant datasource. That can be done thanks to its mechanism to manipulate and populate data on the page, or request data from other web sites, including Google Spreadsheets, Oracle PL/SQL, or other third-party sites that support Google Visualization Query requests. All of that without code changes in our application.

In that way, we can create reports and dashboards as well as analyze and display the data through the wealth of available visualization applications. Moreover, exposes methods to sort, filter, and manipulate data, so we can easily get the information we need.

Next, we are going to see, step by step, how visualizations can be embedded in our applications. Each point will be described with detail and also with a code example about how to do it.
Furthermore, at the end, the whole example about the embedding of a chart using Google Visualization API will be shown.

Thus, the first step consists on adding a HTML tag to the code**.** That element will hold the visualization of the chart. It will be also necessary to pass to the visualization a reference to this element while is being created, so we should assign it an ID, useful to retrieve a reference using the method document.getElementById().

Anything inside the tag will be replaced by the chart when it is drawn.

```
…
<div id="chart_div"></div>
…
```

After that, it will be necessary to load the libraries**.** A visualization drawn with that API requires three libraries to be included or loaded on the page: the Google AJAX API, the Google Visualization API core and a library for each type of visualization. Let's see them with detail:

- Google AJAX API. This one is used to load the Google Visualization API library, through google.load() method, the libraries for each visualization type and also perform other important core tasks such as event handling. That's why must always be loaded first.
  Moreover, it defines also the google.setOnCallback() function that will be useful next.

```
…
<script type="text/javascript" src="http://www.google.com/jsapi"></script>
…
```

- Google Visualization API. This library includes several classes and methods used to create, manipulate and handle visualizations. For example: DataTable object to hold data, Query object for querying data providers, error handlers to help trapping and displaying errors on the page, and so on.
  To load it, must be done in a separate <script> tag after the AJAX API include.

```
<script type=”text/javascript”>
…
google.load('visualization', '1', <<google-package-list>>);
…
</script>
```

where:

‘visualization’	*Loads the Google visualization API.*
‘1’	*Selects the current, stable version of the API.*
<<google-package-list>>	*Specify which Google-authored visualizations to load.*

Is necessary to wait until the Google Visualization API is loaded before continue the rest of the coding. So, to be notified about it, we must call google.setOnloadCallback(callback_handler) after calling google.load(), passing in the name of the function to call when the API is loaded.
The AJAX API will call that function with no parameters when the Google Visualization API and the visualization libraries in google-package-list are loaded.

- Visualization library. Finally, we’ll see how to load a specific visualization (for example, a pie chart or a line chart). Each one is a JavaScript class that exposes methods specific to that visualization, plus a few common methods and events, such as draw() and select.
  Google-authored visualizations are loaded from the Google web site, but is also possible to load third-party ones.

  By one hand, if we want to use the first ones we must list the selected in the <<*google-package-list*>> parameter of the google.load() function. The format should be:

```
…
google.load(‘visualization’,’1’,{'packages':[packages_list]});
…
```

where:

packages_list	*Is a comma-separated list of one or more package names, in quotes.*

On the other hand, to load third-party visualizations we should use a separated <script> include, as the following example shows.

```
…
<script type="text/javascript"
 src="http://www.example.com/third_party_visualization.js"></script>
…
```

Once we've loaded the libraries, the following step consists on prepare the data to visualize it. As we will see, that can be done either specifying the data in code, or querying a remote site for data.

Each visualization stores the data that it displays as a table. In concrete, as a two-dimensional one with rows and columns. Cells are referenced by (row, column) where row is a zero-based row number, and column is either a zero-based column index or a unique ID that can be specified.

A column has up to three properties:

- *Data type*. This one is required. Also, all data in every column must be the same type (null is also allowed as a cell value). The following JavaScript data types are allowed: boolean, string, number and date.
- *ID*. A unique alphanumeric ID that can be used instead of a column index to refer to it. This property is optional.
- *Label*. This property is to offer a user-friendly label for the column. Many charts display the label as chart legend or axis label. Like before, this one is optional.

We should note that different visualizations expect data tables in different format.

Also, data tables are implemented in JavaScript as either a google.visualization.DataTable object or a google.visualization.DataView object (defined in the Google Visualization API). A DataTable is used to create the original data table. In opposite, a DataView is a convenience class that provides a read-only view of a DataTable in which is possible to hide or reorder rows or columns quickly without modifying the underlying data.

Next we're going to see how we can retrieve the data to fill the table and visualize it trough Visualization API in both specified ways.

The first one, is based on define the data to show in the main page. It consists on create the JavaScript DataTable object without any data and populate it by calling addRows() method on it as can be seen in the next example.

```
…

var data = new google.visualization.DataTable();
      // Declare columns and rows
      data.addColumn('type1', 'label1');  // Column 0 with the optional parameter Label.
      data.addColumn('type2', 'label2'); // Column 1with the optional parameter Label.

      // Add data
      data.addRows([
        [value00,value01 ],
        [value10, value11],
        [value20, value21],
        ]);

…
```

At this point we have the table ready to be used for the visualization, so we'll just need to call the methods for create and draw the visualization.

The other way to acquire data for display it consists on requesting data to a data provider. This one is another site that returns a populated DataTable in response to a request from the code.

To send that requests, is necessary to create a Query object and specify a URI for the data source indicating what data is being requested. Some data providers also accept SQL-like query strings to sort or filter the data.

Moreover, it will be necessary to process the query response. For this, we'll need a handler function that will be called when the request returns. The parameter passed in to the handler function is of type google.visualization.QueryResponse. If the request was successful the response contains the data table, else if the request failed, the response contains information about the error. So, it will be necessary to check it by calling the method response.isError(). If there isn't we've the table ready to pass it to the visualization and display it.

The following example shows that process:

```
…

function initialize() {

  var query = new google.visualization.Query('data source URI');

  // Optionally we can manipulate the data with SQL-like strings.
  query.setQuery('SQL-like string');

  // Send the query with a callback function.
  query.send(handleQueryResponse);
}

function handleQueryResponse(response) {
  // Called when the query response is returned.

// Continue on the next page…
```

```
// Cheking if there's an error.

if (response.isError()) {
   alert('Error in query: ' + response.getMessage() + ' ' + response.getDetailedMessage());
   return;
 }

 var data = response.getDataTable();
  ...
}

…
```

At this point, in both ways used to get the table, we have the data ready to be displayed. Thus, the following step will be the creation of a visualization instance. For this, we will call its constructor and passing in a reference to the previously defined element.

```
…

var chart = new google.visualization.ChartType(document.getElementById('chart_div'));

…
```

where:

ChartType *Is the name of the visualization package.*

Finally, we just need to draw the visualization. To do it is just necessary to call draw() method on the visualization and pass in the data to render it on the page. That function takes two parameters: a DataTable or DataView as required, and an options parameter as optional. That second parameter is used to set useful options for the visualization, such as the width, height or colors.

```
…

chart.draw(data, {options});

…
```

where:

options *Is a comma-separated list of pairs option:value.*

We have seen how to display data using Google Visualization API, so let's see now a complete example about it.

```
<html>
…
<script type="text/javascript" src="http://www.google.com/jsapi"></script>
<script type="text/javascript">
google.load("visualization", "1", {packages:["gauge"]});
google.setOnLoadCallback(initialize);

function initialize() {
  // Gathering data from a Spreadsheets document.
  var query = new
google.visualization.Query('http://spreadsheets.google.com/tq?range=A1:B3&headers=-
1&key=pZijkd5BzGvPh769gJ-l9aQ&gid=17&pub=1');

  // Send the query with a callback function.
  Query.send(handleQueryResponse);

}

function handleQueryResponse(response) {
  // Called when the query response is returned.
  Var data = response.getDataTable();
  var chart = new google.visualization.Gauge(document.getElementById('chart_div'));
  var options =  {width: 400, height: 120, redFrom: 90, redTo: 100,
          yellowFrom:75, yellowTo: 90, minorTicks: 5};
  chart.draw(data, options);
}

</script>
</head>

<body>
…
<div id="chart_div"></div>
…
</body>
</html>
```

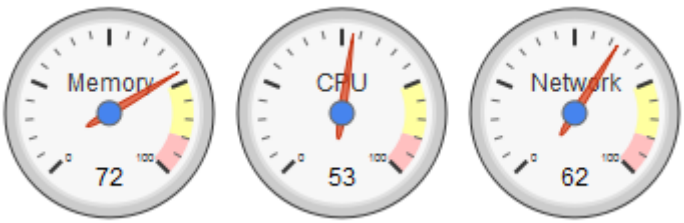
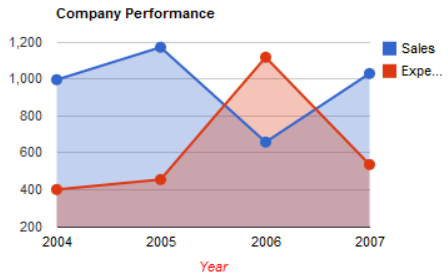
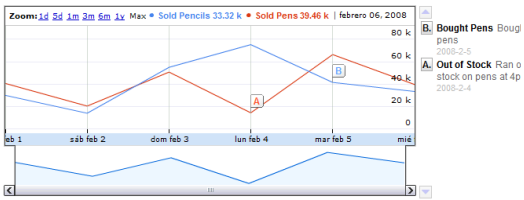
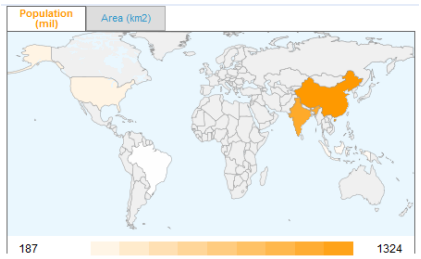
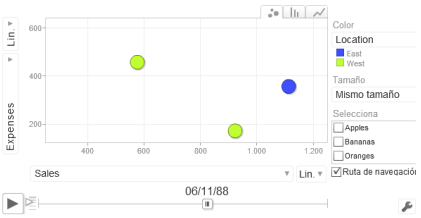
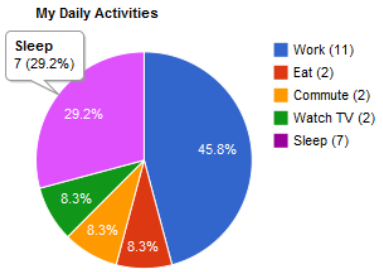
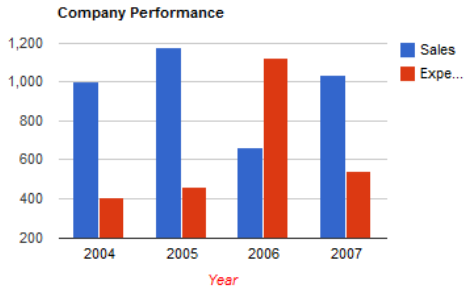


Fig.8 – Returning gauge chart.

Next, before concluding this section, we're going to take a look to some examples about charts that can be used with this API in order to display information (Tab.3). After that, we'll proceed doing a brief comparison about the alternatives concerning the visualization layer we've seen.

Chart type	
Area	
Annotated Time Line	
Intensity Map	
Motion	
Pie	
Column	

Tab.3 – Examples of data displayed as charts using Visualization API.

Finally, for concluding this section dedicated to the visualization layer alternatives offered by Google, we can say that Chart Tools offers us powerful visualization ways to display data as charts in order to help making easier its understanding. Thus, both alternatives are interesting to develop a BI application to help organizations understanding its current situation, and go on making decisions to set the course of themselves.

Now, entering in technical aspects, we are going to compare them. As we've seen, Chart API offers us a really easy and fast way to obtain visualizations just by setting appropriately the parameters of the service's URI. However, its strictness can be its weak point. Even working with POST is possible to do it, is not easy to work with data located in an external datasource, and we require some external methods to gather and parse it to a correct URI format.

On the other side, working with Visualization API, we solve that problem. As is thought to work either with data on the code file either with external data, developers have total freedom to use it not depending on the datasource. Moreover, as allows us to generate interactive charts, using AJAX, we're able to create richer systems, supporting also user interaction.

By using it we lose the fastness that Chart API gives, but we earn quality on our applications.

4.2. Data Layer

In the present section we will see the different options we have to store data in a system developed under Google's infrastructure.

First, we'll focus on the App Engine's[4] Datastore API, seeing either Java version either Python one. For each, we're going to see a description and an explanation about how to manage data trough it.

After that, we will take a look to the options included on Google Visualization API, such as the Data Source Library (Java and Python) and Spreadsheets.

Finally, and for concluding that section, we will see how to use Google Base Data API as a data source.

4.2.1. Datastore API

The Google App Engine Datastore provides robust scalable data storage for web applications. As is part of a tool for develop and host that kind of applications, is designed with these ones in mind, with an emphasis on read and query performance.

In the following lines we will see how it works storing and performing queries over data objects, known as entities.

Thus, an entity has one or more properties, named values, of one of several supported data types, including integers, floating point values, strings, dates, binary data, and so on. A property can be also a reference to another entity.

Each entity also has a key that uniquely identifies it. The simplest key has a kind and a unique ID that can be a numeric value provided by the datastore or a string provided by the application.

An application will be able to fetch an entity from the datastore by using its key, or by performing a query that matches the entity's properties.

These queries can return zero or more entities, and also the results sorted by property values. Moreover, can limit the number of results returned by the datastore to conserve memory and run time.

Another of its characteristics is that is possible to execute multiple operations in a single transaction, and roll it back if any of the operations fail. This feature is especially useful for distributed web applications, where multiple users may be accessing or manipulating the same data object at the same time.

Also, in order to improve the consistency, the datastore replicates all data to multiple storage locations, so if one storage location fails, can switch to another and still access the data. To ensure that the view of the data stays consistent as

[4] The features of the App Engine tool will be explained in the section 4 of this chapter.

it is being updated, an app uses one location as its primary location, and changes to the data on the primary are replicated to the other locations in parallel.

Unlike traditional databases, the Datastore API uses a distributed architecture to manage scaling to very large data sets. An App Engine application can optimize how data is distributed by describing relationships between data objects, and by defining indexes for queries.

So, we can see that is strongly consistent, although it's not a relational database. While its interface has many of the same features of traditional databases, its unique characteristics imply a different way of designing and managing data to take advantage of the ability to scale automatically.

Furthermore, unlike relational model, entities are schemaless: Two entities of the same kind are not obligated to have the same properties, or use the same value types for the same properties. The application is responsible for ensuring that entities conform to a schema when needed.

Once we've seen this description about the Datastore API, we are going to focus on how is possible to manage the described entities with the two versions of the App Engine tool.

4.2.1.1. Datastore Java API

Next, we are going to see how to create and manipulate entities trough Java. The App Engine's Java SDK includes implementations of the Java Data Objects (JDO) and Java Persistence API (JPA)[5], and these will be the interfaces used for modeling and persisting data. For it, these standard interfaces include mechanisms for defining classes for data objects, and for performing queries.

The first one, JDO, works using annotations[6] on Java classes to describe how instances of the class are stored in the datastore as entities, and how entities are recreated as instances when retrieved from the datastore.

Thus, the procedure to create an entity will be the definition of a Java class in first place, with its corresponding methods, and then just will be necessary to import JDO interface tools for including the POJOs that will define how to manage the persistent instances. The following example shows that concept:

[5] To use them we just need to ensure of having its JAR files located in the war/WEB-INF/lib/ directory of the application, and the xml configuration file in the war/WEB-INF/classes/META-INF/ one.
[6] Known as "plain old Java objects" or "POJOs".

Employee.java

```
import com.google.appengine.api.datastore.Key;

import java.util.Date;
import javax.jdo.annotations.IdGeneratorStrategy;
import javax.jdo.annotations.PersistenceCapable;
import javax.jdo.annotations.Persistent;
import javax.jdo.annotations.PrimaryKey;

@PersistenceCapable //Declare the class as capable of being stored and retrieved with JDO.
public class Employee {
    @PrimaryKey //Defining the primary key.
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY) //Defining the primary key.
    private Key key;

    @Persistent //Declare the field able to be stored in the datastore.
    private String firstName;

    @Persistent
    private String lastName;

    @Persistent
    private Date hireDate;

    public Employee(String firstName, String lastName, Date hireDate) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.hireDate = hireDate;
    }



    // Accessors for the fields.  JDO doesn't use these, but the application does.

    public Key getKey() {
        return key;
    }

    public String getFirstName() {
        return firstName;
    }

    // ... other accessors...
}
```

Once we've defined the entity it will be necessary to interact with it, and that will be done trough a PersistenceManager object obtained from a PersistenceManagerFactory object.

Because setting up that kind of object is expensive, we must make sure to get the instance only once in the lifetime of the application, so that can be done wrapping it in a singleton class.

PMF.java

```
import javax.jdo.JDOHelper;
import javax.jdo.PersistenceManagerFactory;

public final class PMF {
    private static final PersistenceManagerFactory pmfInstance =
        JDOHelper.getPersistenceManagerFactory("transactions-optional");[7]

    private PMF() {}

    public static PersistenceManagerFactory get() {
        return pmfInstance;
    }
}
```

Now, it will be possible to instantiate instances of the class and store them in the datastore. We should note that after using the PersistenceManager object will be necessary to close it.

Application code

```
import java.util.Date; import java.util.List;


import javax.jdo.JDOHelper;
import javax.jdo.PersistenceManager;
import javax.jdo.PersistenceManagerFactory;

import Employee;
import PMF;

…
        Employee employee = new Employee("Michael", "Collins", new Date());

        PersistenceManager pm = PMF.get().getPersistenceManager();

        try {
            pm.makePersistent(employee);
        } finally {
            pm.close();
        }
```

[7] "trasactions-optional" parameter refers to the name of the configuration set in the jdoconfig.xml file. If the application uses multiple configuration sets, we'll have to extend this code to call JDOHelper.getPersistenceManagerFactory() as desired.

We can also retrieve instances from the datastore trough the JDOQL[8] query interface included in JDO.

```
…
    String query = "select from " + Employee.class.getName() + " where lastName == 'Collins'";
    List<Employee> employees = (List<Employee>) pm.newQuery(query).execute();

…
```

Moreover, JDO allows us to enrich the defined classes by offering the possibility of define relationships between objects, work with Serializable ones, define inheritance and embed classes.

Thus, in order to create relationships we should proceed in a normal way, creating both classes, and importing the one which will be contained in the container. When the application creates an instance of the container one, will populate also the field referring to the instantiated one, so both entities will be created on the datastore. The key of the contained one will be the key of the container entity as its entity group parent.

Also, JDO allows us to work with inheritance between classes. For this, is just necessary to import the Inheritance and InheritanceStrategy from javax.jdo.annotations, and use the corresponding POJO's to define it.

```
import javax.jdo.annotations.Inheritance;
import javax.jdo.annotations.InheritanceStrategy;
…

@PersistenceCapable
@Inheritance(strategy = InheritanceStrategy.SUBCLASS_TABLE) //setting the class as superclass[9]
public abstract class SuperclassName {
   @PrimaryKey
   @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
   private Key key;

   @Persistent
   private <type> fieldName;
   …
}
```

After that, its subclasses will inherit its properties just by extending from it. So the header of a subclass will look like: *public class SubclassName extends SuperclassName*. In this way, the datastore will save the fields of the superclass as fields of the created entities for the subclasses.

[8] This provided language refers to JDO data classes and fields directly, and includes type checking for query parameters and results. JDOQL is similar to SQL, but is more appropriate for object-oriented databases like the App Engine datastore.

[9] Other inheritance strategies are not supported by App Engine or have an inefficient performance.

On the other hand, if we need our class to contain a Serializable one, we just need to import and label it with the *@Persistent* POJO adding the *(serialized="true")* annotation.

```
import javax.jdo.annotations.Persistent;
import SerializableObjectClassName; //importing the Serializable class of the object we want to add
                                    //to ours.
…

   @Persistent(serialized = "true") //defining it as a Serializable class object able to be stored
   private SerializableObjectClassName attribute_name;
…
```

Finally, embedded annotations allow us to define a relationship like we explained before, without creating a new datastore entity for the contained one. The fields of the contained object will be stored directly in the datastore entity of the container.

```
import javax.jdo.annotations.Embedded;
import javax.jdo.annotations.EmbeddedOnly;
…

@PersistenceCapable
public class ContainerClassName {
   @PrimaryKey
   @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
   Key key;

   @PersistenceCapable
   @EmbeddedOnly //defining the contained class as embeddable one
   public static class ContainedClassName {
      @Persistent
      private <type> fieldName;

     …

      // accessors for the fields
   }

   @Persistent
   @Embedded //embedding it in the container
   private ContainedClassName containedClassFieldName;

…

}
```

Fields of the embedded class will be accessible from the container one. The following code line shows that concept:

```
select from ContainerClassName where containedClassFieldName.fieldName == "value"
```

Next, once we've seen how to work with JDO interface, we are going to take a look the other one given by App Engine to manage data. The most remarkable difference between them is that JDO is focused on object-oriented databases and JPA on relational ones. So, since App Engine's datastore is not a relational database, there are some features of it not supported by the tool's implementation.

Due to its similar way of manage entities like JDO, we will see the most important differences on how to do that.

Thus, instead of interacting with the application trough a PersistenceManager object, in JPA we'll use an EntityManager one obtained from an EntityManagerFactory class in the same way that JDO does.

EMF.java

```
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public final class EMF {
    private static final EntityManagerFactory emfInstance =
        Persistence.createEntityManagerFactory("transactions-optional");

    private EMF() {}

    public static EntityManagerFactory get() {
        return emfInstance;
    }
}
```

The application uses the factory instance to create one EntityManager instance for each request that accesses the datastore.

```
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;

import EMF;

// ...
    EntityManager em = EMF.get().createEntityManager();
```

We will use the EntityManager to store, update and delete data objects, and to perform datastore queries. Like with JDO, when we are done with the EntityManager instance, is necessary to call its close() method.

```
try {
   // ... do stuff with em ...
} finally {
   em.close();
}
```

Moreover, using JPA is also possible to define inheritance between classes in the same way JDO does, although is not possible to obtain instances of the subclasses by querying the superclass.

Finally, JPA also uses annotations for indicate how to manage the persistence of the entity. Thus, we will conclude this section by showing a table with the corresponding POJO for each JPA annotation[10] (Tab.4).

	JDO	***JPA***
Declare the class as persistent	*@PersistenceCapable*	*@Entity*
Field to store	*@Persistent*	*@Basic*
Creating the key	*@PrimaryKey* *@Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)*	*@Id* *@GeneratedValue(strategy = GenerationType.IDENTITY)*
Define inheritance	*@Inheritance(strategy = InheritanceStrategy.SUBCLASS_TABLE)*	*@MappedSuperclass*

Tab.4 – Correspondences between JDO and JPA annotations.

4.2.1.2. Datastore Python API

In the following section, we are going to see how is possible to create and manipulate entities trough Python in the same way we did before.

As is said in the datastore introduction, entities are schemaless, and we need the application to ensure these ones will conform to a schema when needed. For this purpose, the Python SDK includes a rich library of data modeling features that make enforcing a schema easy.

In the Python API, a model describes a kind of entity, including the types and configuration for its properties. An application defines a model using Python classes, with attributes describing the properties. Entities of a kind are represented by instances of the corresponding model class, with instance attributes representing the property values. An entity can be created by calling the constructor of the class, and then stored by calling the put() method as the following example shows.

[10] Importing its corresponding javax.persistence packages, such as .Entity, .GeneratedValue, .Id and so on.

```
import datetime
from google.appengine.ext import db
from google.appengine.api import users

class Employee(db.Model):
    name = db.StringProperty(required=True)
    role = db.StringProperty(required=True, choices=set(["executive", "manager", "producer"]))
    hire_date = db.DateProperty()
    new_hire_training_completed = db.BooleanProperty()
    account = db.UserProperty()

e = Employee(name="",
            role="manager",
            account=users.get_current_user())
e.hire_date = datetime.datetime.now().date()
e.put()
```

Each class attribute is an instance of a subclass of the Property class. A property instance holds configuration for the property, such as whether or not the property is required for the instance to be valid, or a default value to use for the instance if none is provided.

However, sometimes it's useful for an entity to have properties that aren't necessarily like the properties of other entities of the same kind. Such entity is represented in the datastore API by an expando model. Classes of that model subclasses the Expando superclass. Any value assigned to an attribute of an instance of an expando model becomes a property of the datastore entity, using the name of the attribute. These properties are known as dynamic properties. On the other hand, properties defined using Property class instances in class attributes are known as fixed properties.

An expando model can have both fixed and dynamic properties. The model class simply sets class attributes with Property configuration objects for the fixed properties and the application is who creates dynamic properties when it assigns them values.

```
class Person(db.Expando):
    first_name = db.StringProperty()
    last_name = db.StringProperty()
    hobbies = db.StringListProperty()

p = Person(first_name="Thomas", last_name="Anderson")
p.hobbies = ["chess", "travel"]

p.chess_elo_rating = 1350

p.travel_countries_visited = ["Spain", "Italy", "USA", "Brazil"]
p.travel_trip_count = 13
```

As we can see, some of the properties we're assigning value weren't defined in the class Person definition. Because of that, dynamic properties don't have model property definitions, so are not validated. That means that can have a value of any of the datastore base types, including None, and also that two entities of the same kind can have different types of values for the same dynamic property, and one can leave a property unset that the other sets.

The Python API includes another class for data modeling that allows us to define hierarchies of classes, and perform queries that can return entities of a given class or any of its subclasses. Such models and queries are called *polymorphic* because they allow instances of one class to be results for a query of a parent class. The following example shows that concept:

```
from google.appengine.ext import db
from google.appengine.ext.db import polymodel


class Contact(polymodel.PolyModel):

    phone_number = db.PhoneNumberProperty()
    address = db.PostalAddressProperty()


class Person(Contact):

    first_name = db.StringProperty()
    last_name = db.StringProperty()
    mobile_number = db.PhoneNumberProperty()


class Company(Contact):

    name = db.StringProperty()
    fax_number = db.PhoneNumberProperty()
```

Thus, in that way we can create instances of a subclass adding attributes of the superclass, and retrieve instances of the subclass by querying the superclass.

```
p = Person(phone_number='0034-971-82-44-15',
          address='Av.Pataliebre, 45, 08999, Albacete',
          first_name='Marcial',
          last_name='Ruiz',
          mobile_number='0034-660-33-20-15')
p.put()

…

for contact in Contact.all():
    print 'Phone: %s\nAddress: %s\n\n'
          % (contact.phone,
             contact.address))
```

Furthermore, using the datastore Python API is possible to define references between entities. For this, we just need to use the ReferenceProperty class in the following way:

```
class FirstModel(db.Model):
   prop = db.IntegerProperty()


class SecondModel(db.Model):
   reference = db.ReferenceProperty(FirstModel)


obj1 = FirstModel()
obj1.prop = 42
obj1.put()


obj2 = SecondModel()



# Defining the reference value passing the key of the obj1 entity.
obj2.reference = obj1.key()
obj2.put()
…
```

Now is possible to set the value and obtain instances of the FirstModel object through the SecondModel one.

```
obj2.reference.prop = 999
obj2.reference.put()

…

results = db.GqlQuery("SELECT * FROM SecondModel")
another_obj = results.fetch(1)[0]
v = another_obj.reference.prop
```

As we can see in the example above, the API also allows us to obtain instances of entities via query interfaces. In concrete, it provides two of that: a query object interface, and a SQL-like query language called GQL. A query returns entities in the form of instances of the model classes that can be modified and put back into the datastore. Let's see now some examples about it:

```
#example using the query object interface
class Song(db.Model):
   title = db.StringProperty()
   composer = db.StringProperty()
   date = db.DateTimeProperty()


#obtaining all instances of the class Song
query = Song.all()
#continue on the next page…
```

```
#obtaining a concrete instance of the class Song
ancestor = Song.get_by_key_name('my_song')
query = db.Query(ancestor)


…

#example using the GQL query interface

import datetime
from google.appengine.ext import db
from google.appengine.api import users

class Employee(db.Model):
    name = db.StringProperty(required=True)
    role = db.StringProperty(required=True, choices=set(["executive", "manager", "producer"]))
    hire_date = db.DateProperty()
    new_hire_training_completed = db.BooleanProperty()
    account = db.UserProperty()
…

#building a list of employees registered on the training
training_registration_list = [users.User("marcial.ruiz@example.com"),
                              users.User("michaelcollins@example.com"),
                              users.User("budspencer@example.com")]

#obtaining the matching list of employees that appear in the training_registration_list
employees_trained = db.GqlQuery("SELECT * FROM Employee WHERE account IN :1",
                                 training_registration_list)

#modifiying instances and putting them back into the datastore
for e in employees_trained:
    e.new_hire_training_completed = True
    db.put(e)
```

Once we've seen the Datastore API offered by App Engine, we can conclude by saying that this one is an interesting alternative to implement a data layer for some system.

Either working with Java either with Python, we are free to create an entities class structure to model the way to save data as we need, defining relationships, inherence, and so on. Also, the API offers us interfaces and methods to allow us to manage that data easily from the application code.

Furthermore, the fact of having the data layer completely integrated with App Engine by using it is another performance advantage. Through App Engine's administration console we can control the data we've saved inside with some indicators about its usage.

4.2.2. Visualization API data sources

Once we've seen the Datastore API provided by App Engine, we are going to see the options given by Google Visualization API for store data in order to be displayed.

Thus, we are going to take a look first to Data Source Library, seeing either Java version either Python one, and finally we'll see also how Google Spreadsheets can be used for this purpose.

4.2.2.1. Data Source Library

The aim of this library is to make it easy to create a visualization data source. For this, implements the Google Visualization API wire protocol and query language, so we just need to write the code that is required to make the data available to the library in the form of a valid data table for being displayed.

Next, we will see how to proceed with Java and Python versions of the library.

4.2.2.1.1. Data Source Java Library

In this sub-section we'll introduce how manage data using the Java version of Visualization API Data Source library. First, we are going to take a brief view about how it works, and then, a definition on how to create a data source using it.

The simplest implementation of the library involves inheriting from a single class, implementing a member function and running the data source as a Servlet within a Servlet container. The following sequence of events takes place when a visualization queries the data source:

- The Servlet container handles the query and passes it to the data source Java library.
- The library parses the query.
- The implementation code returns a data table to the library.
- The library executes the query on the data table.
- The library renders the data table into the response expected by the visualization.
- The Servlet container returns the response to the visualization.

All that process is illustrated in the following diagram (Fig.9):

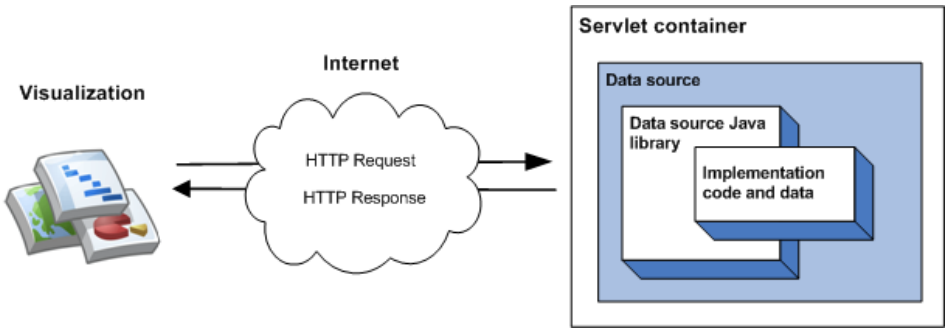


Fig.9 – Management of a visualization query.

On the other hand, if we need to work with larger data sets it will be more likely to use an external data store, such as an external file or database. In that case, the difference we will appreciate between the process described before and this one is that after parsing the query, the implementation code reads the data held in the data store and returns a data table to the library.

If the data set is large, and the data store has querying capabilities, is also possible to optionally use those capabilities in order to increase the efficiency of the data source.

The following diagram (Fig.10) illustrates this alternative implementation:

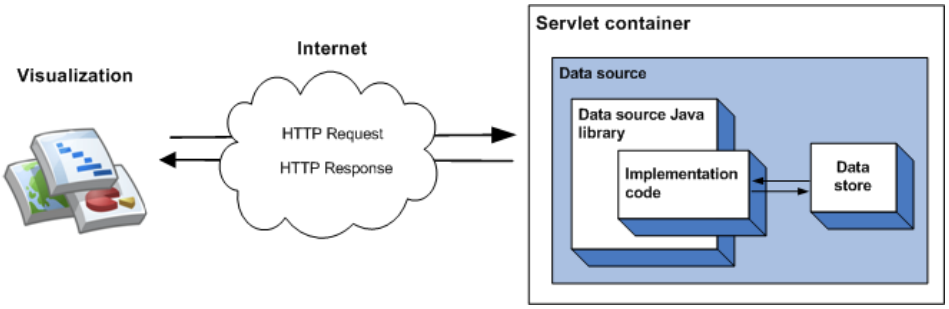


Fig.10 – Management of a visualization query using an external data source.

Next, we will show an exemplified definition about how to implement a data source using the library. First we will see the simple version way, and then the external data store one which will be based on CSV.

Thus, the first we need to do is download the library and include it in the project as the build system requires. Then, we should create a Servlet who will inherit from DataSourceServlet class. It will contain the function generateDataTable() which will be the one who creates, defines and fills the table to pass to the Visualization API. The following example shows it:

```
public class SeminarServlet extends DataSourceServlet {

  @Override
  public DataTable generateDataTable(Query query, HttpServletRequest request) {
    // Create a data table,
    DataTable data = new DataTable();
    ArrayList cd = new ArrayList();
    cd.add(new ColumnDescription("name", ValueType.TEXT, "Name"));

    cd.add(new ColumnDescription("enterprise", ValueType.TEXT, "Enterprise"));
    cd.add(new ColumnDescription("age", ValueType.NUMBER, "Age"));
    cd.add(new ColumnDescription("assistance", ValueType.BOOLEAN, "Assistance confirmed?"));

    data.addColumns(cd);

    // Fill the data table.
    try {
      data.addRowFromValues("Marc Martin", "M.Hari Group", 21, true);
      data.addRowFromValues("Juan Carlos Antunez", "WY Records", 23, true);
      data.addRowFromValues("Jose Reyes", "Gyp S.Y. Services", 24, false);
      data.addRowFromValues("Francisco Javier Torres", "EA",23 , false);
    } catch (TypeMismatchException e) {
      System.out.println("Invalid type!");
    }
    return data;
  }
}
```

The next step we've to do is set correctly the path of the Servlet to define and map it in the server we'll use to run the application. For this, it will be necessary to copy the web.xml file from the library to the server WEB-INF directory, and edit it with the information of the Servlet.

```
<servlet>
  <servlet-name>Seminar</servlet-name>
  <description>Seminar Assistance.</description>
  <servlet-class>SeminarServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>Seminar</servlet-name>
  <url-pattern>/seminar</url-pattern>
</servlet-mapping>
```

Finally, we just need to copy the dependency packages from the library to the WEB-INF/lib server directory. At this point the data source will be able to process any Visualization API query pointing to its URI.

Next, we are going to see how we can create a data source with an external data store based on CSV. Thus, we have to proceed in the same way we did on the description given before: Install the library, create a Servlet and, finally, set the path of it and copy the dependency packages. The only difference will be in the Servlet's code, where instead of define and populate the table directly there,

we will gather the information from a CSV file. The following lines show how to implement it:

```
public class CsvDataSourceServlet extends DataSourceServlet {


  private static final Log log = LogFactory.getLog(CsvDataSourceServlet.class.getName());

  //That parameter will contain the URI of the CSV to load.
  private static final String URL_PARAM_NAME = "url";

  @Override
  public DataTable generateDataTable(Query query, HttpServletRequest request)
      throws DataSourceException {
    String url = request.getParameter(URL_PARAM_NAME);
    if (StringUtils.isEmpty(url)) {
      log.error("url parameter not provided.");
      throw new DataSourceException(ReasonType.INVALID_REQUEST, "url parameter not
provided");
    }

    Reader reader;
    try {
      reader = new BufferedReader(new InputStreamReader(new URL(url).openStream()));
    } catch (MalformedURLException e) {
      log.error("url is malformed: " + url);
      throw new DataSourceException(ReasonType.INVALID_REQUEST, "url is malformed: " + url);
    } catch (IOException e) {
      log.error("Couldn't read from url: " + url, e);
      throw new DataSourceException(ReasonType.INVALID_REQUEST, "Couldn't read from url: " +
url);
    }
    DataTable dataTable = null;
    ULocale requestLocale = DataSourceHelper.getLocaleFromRequest(request);
    try {
      dataTable = CsvDataSourceHelper.read(reader, null, true, requestLocale);
    } catch (IOException e) {
      log.error("Couldn't read from url: " + url, e);
      throw new DataSourceException(ReasonType.INVALID_REQUEST, "Couldn't read from url: " +
url);
    }
    return dataTable;
  }
}
```

So, as we can see, the Servlet obtains the direction of the CSV file as a request parameter, and then proceeds to read its information to fill the returning DataTable object.

Finally, at this point, the created data source is totally functional. Assuming that we've mapped the Servlet's path as /csv in the web.xml file, we'll just need to pass its URI (with the CSV file's URI associated as a Servlet parameter) to the Visualization API query method.

```
query = new google.visualization.Query('csv?url=http://..csv_uri…/csv_file_name.csv');
```

4.2.2.1.2. Data Source Python Library

As we've seen, Visualization API allows us to create data sources for store data in order to be displayed. Now, we are going to see how this can be done using the Pyhton library.

The aim of that library is to create DataTable objects in Python to be consumed by visualizations. It will allow us to generate outputs in three different formats:

- JSON string, to pass it into a DataTable constructor to populate it.
- JSON response string, to be returned in response to a data request. Useful if we're acting just as a data source for external visualizations.
- JavaScript string, that consists on several lines of JavaScript code that will create and populate a google.visualization.DataTable object with the data from the Python table. Typically used for debugging only.

Now we will see a description about how we can use the library to create a data source. First of all, we need to import it and instantiate the gviz_api.DataTable class. That class takes two parameters: a table schema, which will describe the format of the data in the table, and optional data to populate the table with. Is also possible to add data later or completely overwrite the data, but not remove individual rows or clear out the table schema.

The next step will consist on define the table schema. This one is specified by the table_description parameter passed into the constructor. As we said, is not possible to change it later. The schema describes all the columns in the table: the data type of each column, the ID, and an optional label.

Each column is described by a tuple in that way:

(ID [,data_type [,label [,custom_properties]]])

where

ID	*Is a string ID used to identify the column, so must be unique.*
data_type	*Is an optional string descriptor of the Python data type of the data in that column.*
Label	*Acts as a user-friendly name for the column, which might be displayed as part of the visualization. If not specified, the ID value is used.*
custom_properties	*Is a {String:String} dictionary of custom column properties.*

So as a result, the table schema is a collection of column descriptor tuples. Every list member, dictionary key or dictionary value must be either another collection or a descriptor tuple. Is also possible to use any combination of dictionaries or lists, but every key, value or member must eventually evaluate to a descriptor tuple. Here are some examples:

List of columns: [('a', 'number'), ('b', 'string')]
Dictionary of lists: {('a', 'number'): [('b', 'number'), ('c', 'string')]}
Dictionary of dictionaries: {('a', 'number'): {'b': 'number', 'c': 'string'}}
And so on, with any level of nesting.

Once we've defined the table schema it will be necessary to populate it with data. In order to add it to the table, we must build a structure of data elements in the exact same structure as the defined by the table schema. So, for example, if our schema is a list, the data must be a list, or if the schema is a dictionary, data must be a dictionary.

```
schema: [("color", "string"), ("shape", "string")]
data: [["blue", "square"], ["red", "circle"]]
schema: {("rowname", "string"): [("color", "string"), ("shape", "string")] }
data: {"row1": ["blue", "square"], "row2": ["red", "circle"]}
```

Finally we'll just need to generate the data output. The most common of the three formats we mentioned before is JSON, so it will be necessary to use the ToJsonResponse() function to create the data to return. However, it will be possible to call any of the other output methods to return other formats, including comma-separated values, tab-separated values, and JavaScript.

Now, for concluding this section we are going to see an example about how to use the JSON and JavaScript output formats.

```
#!/usr/bin/python

import gviz_api #importing the library

page_template = """
<html>
  <script src="http://www.google.com/jsapi" type="text/javascript"></script>
  <script>
    google.load('visualization', '1', {packages:['table']});

    google.setOnLoadCallback(drawTable); #the callback function will draw the table in both formats
    function drawTable() {
      %(jscode)s
      var jscode_table = new
google.visualization.Table(document.getElementById('table_div_jscode'));
      jscode_table.draw(jscode_data, {showRowNumber: true});

      var json_table = new google.visualization.Table(document.getElementById('table_div_json'));
      var json_data = new google.visualization.DataTable(%(json)s, 0.6);
      json_table.draw(json_data, {showRowNumber: true});
    }
  </script>
  <body>

    <H1>Table created using ToJSCode</H1>
    <div id="table_div_jscode"></div>
    <H1>Table created using ToJSon</H1>
    <div id="table_div_json"></div>
  </body>
</html>
"""

#Continue on the following page…
```

```
def main():
  # Creating the schema and the data
  description = {"name": ("string", "Name"),
                 "salary": ("number", "Salary"),
                 "full_time": ("boolean", "Full Time Employee")}
  data = [{"name": "Mike", "salary": (10000, "$10,000"), "full_time": True},
          {"name": "Jim", "salary": (800, "$800"), "full_time": False},
          {"name": "Alice", "salary": (12500, "$12,500"), "full_time": True},
          {"name": "Bob", "salary": (7000, "$7,000"), "full_time": True}]

  # Loading it into gviz_api.DataTable
  data_table = gviz_api.DataTable(description)
  data_table.LoadData(data)

  # Creating a JavaScript code string
  jscode = data_table.ToJSCode("jscode_data",
                               columns_order=("name", "salary", "full_time"),
                               order_by="salary")
  # Creating a JSon string
  json = data_table.ToJSon(columns_order=("name", "salary", "full_time"),
                           order_by="salary")

  # Putting the JavaScript code and JSon string into the template
  print "Content-type: text/html"
  print
  print page_template % vars()


if __name__ == '__main__':
  main()
```

If instead of populate the table and output it in the same file we want to work dynamically acting as a data source for a visualization we should work with JSON Response format. For our purpose will be more interesting to proceed in that way, so now we're going to see an example about how to do it.

We just need to create a Python file with the following content:

```
#!/usr/bin/python

import gviz_api #loading the library

#defining the schema and the data
description = {"name": ("string", "Name"),
               "salary": ("number", "Salary"),
               "full_time": ("boolean", "Full Time Employee")}
data = [{"name": "Mike", "salary": (10000, "$10,000"), "full_time": True},
        {"name": "Jim", "salary": (800, "$800"), "full_time": False},
        {"name": "Alice", "salary": (12500, "$12,500"), "full_time": True},
        {"name": "Bob", "salary": (7000, "$7,000"), "full_time": True}]

#populating the DataTable object
data_table = gviz_api.DataTable(description)
data_table.LoadData(data)

print "Content-type: text/plain"
print
print data_table.ToJSonResponse(columns_order=("name", "salary", "full_time"),
                                order_by="salary")
```

Finally, we'll just need to set the URI of the data source in our Visualization API query string.

Once we've seen the Java and Python Data Source library, we can say that its main advantage is that they're fully compatibility with Visualization API and, in consequence, with App Engine.

Anyway, its strictness to define a data layer as we want can be seen as a weakness.

4.2.2.2. Google Spreadsheets

Once we've seen the Google Visualization libraries for create data sources with Java and Python, we're going to see how we can build a data source using Google Spreadsheets documents.

The most important characteristic of using a spreadsheet as a data source is its simplicity of management. We just need to find out the proper URI to use for a sheet or a range of cells in the document to which it has access, and using that URI as a data source URI in visualization queries. With it, we'll be able to embed the results as gadgets or as non-gadgetized ones.

Moreover, Google Spreadsheets support the Google Visualization API query language for sorting and filtering data, so we'll be also able to easily manipulate the data as we need.

Next, we are going to describe how we can use it as a data source for store data and access there when we need to display it.

The first thing we should do is to open the document where we have the data we need to display. This spreadsheet must have exactly the format expected by the concrete visualization we want to use, and it should have viewing privileges set properly. The simplest way is to set that ones for everyone, but is also possible to restrict view privileges to people who will use that spreadsheet as a data source.

The next step will be select the cells range to visualize, and obtain the URI of them. For this, we should click 'Insert' option from the spreadsheet toolbar, and choose 'Gadget'. Then, we just need to select any of the gadget offered types.

Once we've it inserted in the document, we must open the gadget's menu by clicking the arrow in the title bar, and select the option 'Get query data source url…'. This will display our desired URI, which will look something similar like:

http://spreadsheets.google.com?...&key=123ABC&range=B10:B22

At this moment we've the address of our data source, and we can use it to set the Visualization API query parameter to display the information.

The query also supports the following optional parameters:

- headers=N. Specifies how many rows are header rows, where N is an integer zero or greater. These will be excluded from the data and assigned as column labels in the data table. If we don't specify this parameter, the spreadsheet will guess how many rows are header rows. Should be noted that if all columns are string data, the spreadsheet might have difficulty determining which rows are header rows without this parameter.
- gid=N. This parameter specifies which sheet in a multi-sheet document to link to, if we are not linking to the first sheet. N is the sheet's ID number, an integer zero or greater. It's one less than the number in the sheet name when it is created: for example, gid=0 for Sheet1.
- sheet=sheet_name. Like the parameter explained above, is useful to specify which sheet in a multi-sheet document we are linking to, if we are not linking to the first sheet. Instead of using the sheet ID number, sheet_name refers to the display name of that sheet.

Thus, queries will look similar to something like:

```
var query = new
google.visualization.Query('http://spreadsheets.google.com/tq?range=A1:C6&headers=1&key=pZij
kd5BzGvPh769gJ-l9aQ&gid=17');
```

Once we get the returned data we can proceed to work with it as is usual with Google Visualization API.

As we've seen, the using of Google Spreadsheets as a data source for display data as charts using Visualization API is a really easy way to get the expected data in the correct format and efficiently.

4.2.3. Google Base Data API

In this section, we are going to see how we can develop applications using an xml-like database. This will be possible trough Google Base Data API.

First of all, in order to contextualize, we will introduce the Google Base tool. Thus, we can define it as a service for submiting all kinds of content for Google to host and to make searchable online. It allows content providers to upload structured data, manage it, surface it across Google search properties, and syndicate it via APIs, gadgets and gadget ads. Using the Google Base Data API, developers can programmatically access Google Base.

Once we've seen that brief explanation about the service, we are going to show how its Data API works.

Each Google Base entry is defined by a Data Item. These ones are xml constructs that describes the content using a set of attributes in a typed name/value pairs. Trough Base Data API we are able to search for that Data Items, discover its metadata, and also manage them by inserting, updating or deleting via webservice[11].

The Google Base data API provides two feeds for data:

- snippets feed (feeds/snippets). A read-only feed that is used to access all published items in Google Base. This feed does not require authentication, and allows us to perform attribute and full text queries on it.
- items feed (feeds/items/). A read/write, customer-specific feed that can be used to insert, update, delete, and query our own data. This feed requires authentication.

So we will execute queries against them depending on the scope of the data we need to retrieve. As for our purpose is more interesting to be able to manage data, we will focus on the items feed.

Google Base items use two kinds of namespaces, the Atom and Google Base ones. Attributes like <title>, <link>, <entry>, <id>, and <category> are encoded as XML elements in the Atom namespace. All other attributes use the Google Base namespace. This one is identified by the URI http://base.google.com/ns/1.0.

Also, the gm namespace is used to represent metadata, and it's identified by the URL http://base.google.com/ns-metadata/1.0.

The following example shows how looks like an item entry:

```
<?xml version='1.0'?>
<entry xmlns='http://www.w3.org/2005/Atom'
   xmlns:g='http://base.google.com/ns/1.0'>
   <author>
     <name>name</name>
     <email>name@gmail.com</email>
   </author>
   <category scheme='http://www.google.com/type' term='googlebase.item'/>
   <title type='text'>He Jingxian's chicken</title>
   <content type='xhtml'>
     <div xmlns='http://www.w3.org/1999/xhtml'>Delectable Sichuan specialty</div>
   </content>
   <link rel='alternate' type='text/html' href='http://www.host.com/123456jsh9'/>
   <g:item_type>Recipes</g:item_type>
   <g:cooking_time>30</g:cooking_time>
   <g:main_ingredient>chicken</g:main_ingredient>
   <g:main_ingredient>chili peppers</g:main_ingredient>
   <g:main_ingredient>peanuts</g:main_ingredient>
   <g:spices_used>Szechuan peppercorn</g:spices_used>
   <g:cook_technique>blackened</g:cook_technique>
</entry>
```

[11] We should note that all data submitted to Google Base is public, except for any attributes that we explicitly hide. On the following lines will be explained how we need to proceed to not allow everybody access it.

Is possible to optionally include a <g:id> element[12] to uniquely identify our own items. If included, Google Base will consider this attribute value as a surrogate ID for it.

In order to control whether attributes are visible we need specifying the XML attribute access="private". For instance, if we include the following attribute definition in an entry:

```
<g:ourCustomAttribute type="text" access="private">
  this will not be visible by others
</g:ourCustomAttribute>
```

then the concrete element will not appear on the snippets feed, and will consequently not be visible by other users. It will, however, be visible on the items feed, so that we can access it for queries or updates. The only restriction we have is to only mark our custom attributes as private. All Google-recommended attributes are public and we should not try to make them private, or our items will fail validation.

On the other hand, if we want to hide a whole item, and prevent it to be searchable through various Google search applications or APIs, we just need to set the value of the no_api_syndication attribute as true.

```
<g:no_api_syndication>true<g:no_api_syndication>
```

In the following lines we are going to see how is possible to create items to be hosted and how can we manage and query them from an application. But first of all, will be necessary to obtain an API key and a token to be authenticated.

The first one is possible to get it when we register in Google Base with the developer's Google Account. We'll need to supply this API key every time we make a request to the Google Base data API.

After that, in order to obtain the token to authenticate users, we can proceed via Programmatic Login or via AuthSub. Both require a username and password of a Google account.

If we are using the Programmatic Login system, the desktop client will ask the user for their credentials, and then sends those credentials to the Google authentication system. If authentication succeeds, then the system returns a token that the client subsequently uses in an HTTP Authorization header when it sends GData requests.

If instead of that we are working with AuthSub, it will be necessary to log in to a page that provides our Google username and password to a Google authentication server. The AuthSub processing creates a token that is then passed back to be used to authenticate us whenever we perform an action (insert, update, delete) from our application. So after being redirected to our

[12] Also known as SKUs.

application, the token will be included in the header of subsequent http requests.

Once we've seen how to proceed registering, we'll take a look on how is possible to work with it.

Our client application can connect to the Google Base API server using an Atom parser, such as Rome, or a command-line tool, such as curl coupled with an XML parser.

Thus, in order to manage data items we'll need our language to have an XML parser and html request operations.

The first thing we'll need to do is specify version 2 of the API. That can be done by setting the GData-Version HTTP header:

GData-Version: 2

In order to access the itemtypes feed or attributes feed we'll need to proceed by making GET requests to their URIs, setting also its parameters to filter. To do that we can proceed via three ways:

- Full-text queries: executed using the q parameter in query URIs
  q=text query
- Structured queries: using the bq parameter in query URIs
  bq=text query [attrib_name(attrib_type):value] [attrib_name(attrib_type):value] ...
- Attribute queries: using the syntax *[<attribute name>: <attribute value>]*

So, a query for an item will look like:

```
GET /base/feeds/items?query_parameters
Content-Type: application/atom+xml
Authorization: AuthSub token="concrete_token"
X-Google-Key: key=key_value
```

As a response we'll obtain an xml file with the results, so we just need to parse it in order to obtain the desired results.
If we want to insert an item we'll proceed in the same way, but instead of sending a GET request we'll do it by a POST one, including the xml code of our item.

```
POST /base/feeds/items
Content-Type: application/atom+xml
Authorization: AuthSub token="concrete_token"
X-Google-Key: key=key_value

<?xml version='1.0'?>
<entry xmlns='http://www.w3.org/2005/Atom'
 xmlns:g='http://base.google.com/ns/1.0'>

…

</entry>
```

Finally if we need to update or delete an item, we'll need to use http PUT or DELETE respectively indicating the concrete item URI.

```
PUT /base/feeds/items/itemID
Content-Type: application/atom+xml
Authorization: AuthSub token="concrete_token"
X-Google-Key: key=key_value

<?xml version='1.0'?>
<entry xmlns='http://www.w3.org/2005/Atom'
 xmlns:g='http://base.google.com/ns/1.0'>

…new xml code of the item to update…

</entry>
```

…

```
DELETE /base/feeds/items/itemID
Content-Type: application/atom+xml
Authorization: AuthSub token="concrete_token"
X-Google-Key: key=key_value
```

At this point we've seen to work managing data items. Parsing the answers of the queries we'll be able to access to the desired information in order to format it correctly to be displayed as a chart.

We should note that Google Base Data API allows also us to work with Google Data Java/Python client libraries in order to ease the described process.

Finally, for concluding this section, we can say that this API is really interesting if we want to use xml in our applications. Working with the items feed we can obtain the advantage of having a data layer totally adaptable to our needs, defining exactly what we want to store, and independent on the core's platform.

However, as is not thought with App Engine and some alternative to display data as charts in mind could be seen as too much independent, so we would need some extra coding to connect it appropriately.

4.3. Others

Once we have seen the different alternatives classified in terms of their typology we are going to take a brief view over some other useful tools, offered by Google, in order to build BI applications.

As there are just complements to add to a developed system in order to improve it, we'll not enter in specific aspects of them and just will be introduced.

Thus, we'll see the Secure Data Connector interface, the Google Web Toolkit and, finally, the Google Visualization Toolbar.

4.3.1. Secure Data Connector

This client tool that is possible to use in combination with Google Apps allows us to connect applications and spreadsheets to data that is protected by a corporate firewall.

SDC[13] forms an encrypted connection between the enterprise data and Google Apps, allowing us to control who in our domain can access which resources using that suite of productivity tools. Thus, using it is possible to build private gadgets, spreadsheets and applications that interact with the existing corporate systems.

The following illustration (Fig.11) shows SDC connection components:

[13] Since now we will refer to Secure Data Connector as SDC.

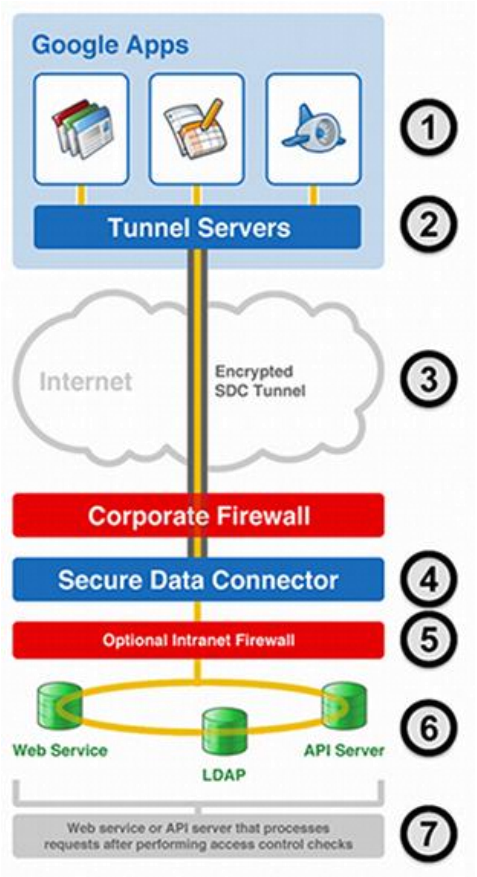


Fig.11 – Components of a solution using SDC.

The diagram steps are:

1. Google Apps forwards authorized data requests from users who are within the Google Apps domain to the Google tunnel protocol servers.
2. The tunnel servers validate that a user is authorized to make the request to the specified resource. These servers are connected by an encrypted tunnel to SDC, which runs within a company's internal network.
3. The tunnel protocol allows SDC to connect to a Google tunnel server, authenticate, and encrypt the data that flows across the Internet.
4. SDC uses resource rules to validate if a user is authorized to make a request to a specified resource.
5. An optional intranet firewall can be used to provide extra network security.
6. SDC performs a network request to the specified resource or services.
7. The service validates the signed request, checks the credentials, and if the user is authorized, returns the data.

Thus, SDC provides us a secure link between Google Apps and corporation's data, encrypting it connection, and acts also as a filter to decide which services can interact with some internal systems of the company. In conclusion, we can say that is a useful tool to add another security level to systems developed using Google Apps infrastructure.

4.3.2. Google Web Toolkit

This toolkit[14] allows us to create AJAX applications in Java programming language that will be later compiled by GWT in optimized executable JavaScript code that works automatically in the most of browsers.

Thanks to it, during the development of an application, it will be possible to quickly repeat the typical "edit - update - see" JavaScript cycle and take advantage of being able to debug and analyze one by one all the Java code lines. Thus, we would immediately see the changes we made in the Java code trough the hosted mode GWT browser, instead of recompile all in order to see it via server.

Also, the AJAX code will be compiled as plain JavaScript, however while we will be developing the application it will run on the Java virtual machine as bytecode.

Moreover, when we will be ready for implementation, GWT will compile Java source code into optimized JavaScript separate files which will be able to be used by any web-server. That will have repercussion allowing the application to be executed in any web browser without need of detecting which is the one used by the user.

Thus, GWT allows us to easily create complete applications using AJAX and take advantage of its characteristics, creating user-friendly interfaces.

4.3.3. Google Visualization Toolbar

In this section we are going to take a look to a complement included by Google Visualization API named Google Visualization Toolbar. This one allows users to export the underlying data of each chart into a CSV file, an HTML table, or to provide code to embed the visualization into an arbitrary web page or gadget.

We should note that it will be just possible to add it on those charts which its data come from a data source referenced by a URI.

The process for using it is simple. We just need to add a few lines including the google.visualization.drawToolbar() method into the visualization's code as the following example shows:

[14] Since now we will refer to it as GWT.

```
<html>
…
<script type="text/javascript" src="http://www.google.com/jsapi"></script>
<script type="text/javascript">
google.load("visualization", "1", {packages:["gauge"]});
// The callback will refer to a new function used to call the creation of the visualization and its
toolbar
google.setOnLoadCallback(drawVisAndToolbar);

function drawVisAndToolbar(){
   // first we call the function to gather the data for the visualization as is defined in the previous
section
  initialize();
  // calling the function to include the toolbar
  drawToolbar();
}
…
// drawing the visualization…
…

function drawToolbar() {
  // including the components to the toolbar
 var components = [
     {type: 'igoogle', datasource: 'datasource URI',
      gadget: 'gadget's xml definition URI string'},
     {type: 'html', datasource: datasource URI'},
     {type: 'csv', datasource: 'datasource URI'},
     {type: 'htmlcode', datasource: 'datasource URI',
      gadget: 'gadget's xml definition URI string'}
  ];

  var container = document.getElementById('toolbar_div');
  google.visualization.drawToolbar(container, components);
};

</script>
</head>

<body>
…
<div id="chart_div"></div>
<div id="toolbar_div"></div>
…
</body>
</html>
```

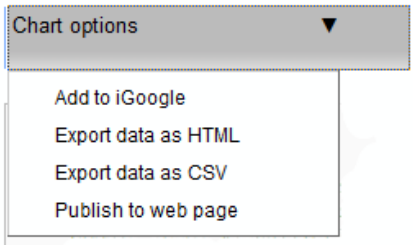


Fig.12 – Resulting Toolbar for some visualization.

As a result, we offer to the user the possibility of easily manage the visualization's data in a different way. So, in conclusion, we can say that this complement is really useful in order to improve the features of a system developed using Visualization API.

4.4. App Engine

Until now, we have seen the different alternatives offered to store and display data, but we still need something to connect both parts and who can also act as a filter to transform stored data into information to be shown.

That's why in this point of the document we'll talk about the most important part of each BI solution: its core. For it purpose, Google Apps offers us the App Engine tool.

As is said in the previous chapter, Google Apps is more than a productivity service. Despite being a suite of tools with the aim of improve the corporation's performance, it also allows developers to build, host and run its applications under Google's infrastructure with its corresponding advantages, and that can be done through App Engine.

So, in our own BI system it will act as a link between data and its visualization, collecting it from its different sources, debugging and finally displaying it as useful information for a successful company management.

Thanks to its power it will let also control the whole processes involved by the system, such as authentication and user requests processing.

In the next chapter, '5. The development of easyBIew', will be explained and exemplified how to use App Engine for this purpose, but first, we'll introduce the tool taking a look to its characteristics.

4.4.1. Application Environment

In this section we'll talk about the App Engine's environment for developed applications. First, we will make a brief introduction about it where will see some of its features, and after that, a part by part description.

Thus, App Engine allows us to develop easily applications running reliably, even with heavy workloads and large amounts of data. Will be also easy to maintain and update when traffic and data storage needs increase.

Moreover, thanks to the offered runtime environments these applications can be written in various programming languages. Trough them, we will also ensure a quickly and safely run, without interference from other applications into the system.

Let's take a look now into some of its features:

- Includes a web server fully compatible with common web technologies.
- Permanent storage, with query, order and transaction functions, automatic scaling and load balancing.
- User authentication and mail sending APIs trough Google accounts.
- A complete local development environment that simulates App Engine on local computers.

- Events to activate scheduled tasks at specified times and at regular intervals.

Now, for continuing the presentation of App Engine's application environment we will see a part by part description of it.

For this, we are going to take a look first to the sandbox and then to the offered runtime environments. Moreover, we will talk about the offered services, integration of developed applications with Google accounts and, finally, about scheduled jobs[15].

The sandbox

Systems build with App Engine run in a secure environment that provides limited access to the underlying operating system. These limitations allow App Engine to distribute web applications on multiple servers and start and stop them also as traffic demands. The sandbox isolates the application in its own trust secure environment, completely independent of hardware, operating system and web server's physical location.

Some examples of the sandbox safe environment limitations are:

- An application can only access other computers on the Internet through e-mail and URI extraction services provided. Other hosts can only be connected to the application using HTTP (or HTTPS) in the standard ports.

- An application can't write to the file system. It can read files, but only those uploaded to the application code. So, for store the necessary data between requests the application must use App Engine's storage system, Memcache or other services.

- Application code only runs in response to a web request or a cron job[16] and must return response data within a period of 30 seconds in any case. A requests controller may not generate a thread or execute code after sending the response.

[15] The description about the App Engine's data storage system can be found in the section 4.2.1. of this chapter.
[16] Explained in the 'Scheduled jobs' paragraph of this section.

Runtime environtments

Applications can be executed under both runtime environments offered by the tool: Java and Python ones. Both provide standard protocols and common technologies for web applications development. In the following lines we will see the characteristics of each.

First, we are going to talk about the Java one. Thanks to it is possible to develop applications using common Java web development tools and API standards. These applications will interact with the environment using the Java Servlet standard, and can also use common web application technologies such as JavaServer Pages (JSPs).

Moreover, applications can access most App Engine services using Java standard APIs, such as JDO, JPA, JavaMail, Java.net HTTP and so on.

The restrictions of the sandbox environment are implemented in the JVM, so an application can use any JVM bytecode or library feature, as long as it does not exceed the sandbox restrictions we saw some lines above. For instance, bytecode that attempts to open a socket or write to a file will throw a runtime exception.

Furthermore, with the use of JVM-compatible compilers or interpreters, is also possible to use other languages to develop web applications, such as JavaScript, Ruby, or Scala.

On the other hand we can find the Python environment. With it is possible to implement applications using the Python programming language, and run it on an optimized Python interpreter. App Engine includes rich APIs and tools for Python web application development, including a detailed data modeling API, a web application framework, and tools for managing and accessing the data.

Moreover, is also possible to take advantage of a wide variety of mature libraries and frameworks for Python web application development, such as Django. However, like in Java, not all of the library's features can run in the sandbox environment because of its constraints.

Another of its characteristics is that provides rich Python APIs for the datastore, Google Accounts, URL fetch, and email services. App Engine also provides a simple Python web application framework called webapp to make it easy to start building applications. We should note that application codes written for the Python environment must be written exclusively in Python, so extensions written in the C language are not supported.

Google accounts integration

App Engine supports integrating an application with Google Accounts for user authentication. Users will be able to sign in with a Google account, and access the email address and displayable name associated with it. Also, one of the advantages of using Google Accounts is that we will let the user start using the

application faster, because may not need to create a new account. It also saves the effort of implementing a user account system just for our application.

Moreover, applications running under Google Apps can use the same features with members of the organization and Google Apps accounts.

The Users API can also tell the application whether the current user is a registered administrator for the application, making it easy to implement administration-only areas.

Services

App Engine provides also a variety of services that enable us to perform common operations when managing applications. These services are accessible via some APIs such as:

- URL Fetch. To access resources on the internet.
- Mail API. To send e-mail messages using Google's infrastructure.
- Memcaché. To provide high-speed accessibility to data.
- Image manipulation. To easily manipulate JPEG and PNG images.

Scheduled jobs

An application running under App Engine can perform tasks outside of responding to web requests. These tasks are also known as "cron jobs", handled by the Cron service, and can be scheduled as we configure, such as on a daily or hourly basis, or also added to a queue by the application itself, such as a background task created while handling a request.

It works by calling a specified URI of the application[17] which will perform the desired task. Either with Java either with Python is possible to use it.

Once we have seen a description about the different parts of the application environment let's take a look to its workflow development.

4.4.2. Workflow development

In order to develop applications to run under Google infrastructure, App Engine offers us software development kits for Java and Python. Each includes a web server application that emulates all of the App Engine services on local machines, and all of the APIs and libraries available on App Engine. The web server also simulates the secure sandbox environment, including checks for attempts to access system resources disallowed in the App Engine runtime environment.

[17] The access to that URI's can be restricted to normal users of the application.

Also, both SDK include a tool to upload applications to App Engine when the code is created. In order to ensure the security for that process the tool prompts us for our Google account email address and password.

If we build a new major release of an application that is already running on App Engine, is possible to upload it as a new version. The old one will continue serving users until we decide to switch to the new version. So, App Engine allows us to test the new version while the old one is still running.

The Java SDK runs on any platform with Java 5 or Java 6, and is available for download it as a Zip file. If we use the Eclipse development environment, is also possible to use the Google Plugin for Eclipse to create, test and upload App Engine applications. Moreover, the SDK includes command-line tools for running the development server and uploading applications.

On the other hand, the Python SDK is implemented in pure Python, and runs on any platform with Python 2.5, including Windows, Mac OS X and Linux. Its download is also available as a Zip file, and installers are available for Windows and Mac OS X.

Finally, App Engine offers us a web-based Administration Console to manage our uploaded applications. Is possible to use it to create new applications, configure domain names, change which version of the application is live, administrate developer accounts, examine access and error logs, and browse an application's datastore.

In conclusion we can say that App Engine offers us a rich environment to build and run multi-platform applications. As we have seen, due to its characteristics the development of web-based systems becomes easier, maintaining at the same time all the principles that a software tool should has.

Thanks to its adaptability for work with the different APIs that Google Code offers to developers, we can manage the totality of a system, deciding how to implement each part of it and obtaining powerful tools as a result.

4.5. Building a BI solution

At this moment, we have seen the different tools and services that Google Code offers for develop applications according to our purpose. What we are going to see now is an overview about how these alternatives must be used and assembled with App Engine in order to obtain a BI solution. As a result we will have a brief guide about the directives we should follow to develop that kind of systems using Google products.

We have seen in the second chapter of the document that a BI solution should have tools to present the information, analyze and forecast it. Google doesn't offer us proper tools to analyze and foresee data, however gives us the possibility to add these kinds of tools without impact on the entire system. Thus, solutions developed using Google will be focused on display the data of the company as graphical information in order to help knowing the situation of it to take better decisions.

In the following lines we are going to see how assemble all described parts to have a complete solution. For this, we will see first a diagram about the architecture schema of a system, and then we will explain it part by part.

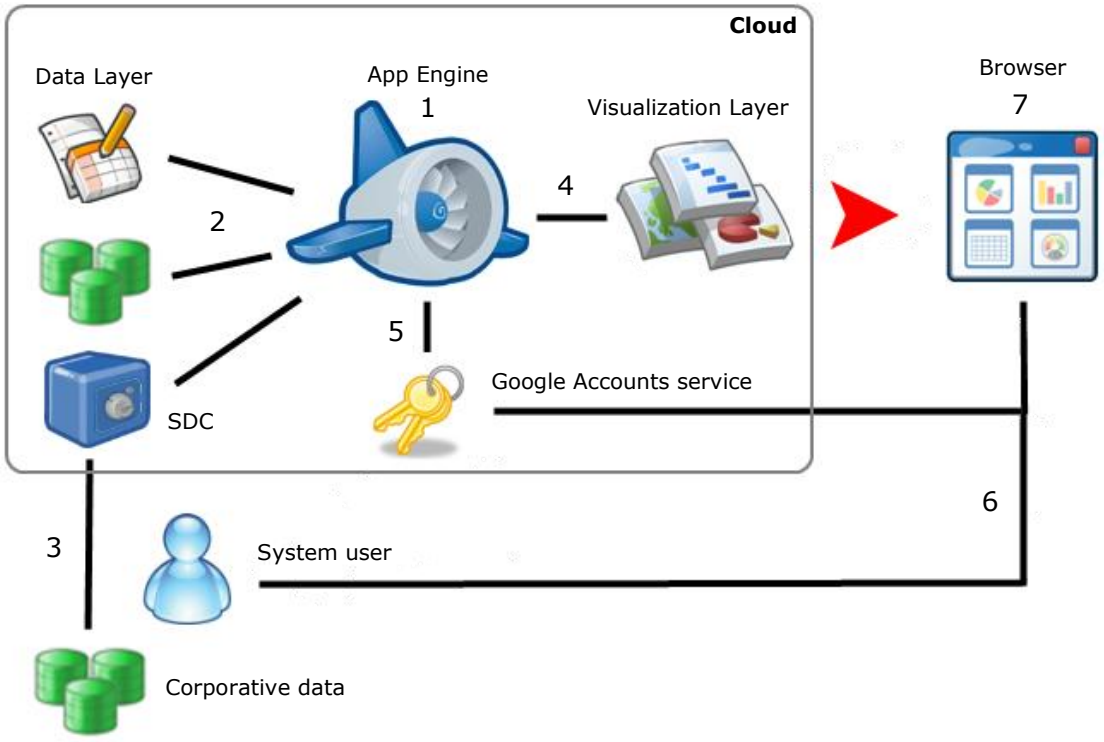


Fig.13 – Architecture of a BI solution developed with Google.

1. App Engine. As we've said, App Engine will act as the core of the application, holding the files, libraries and classes to build it, processing user requests, retrieving data from the data source and displaying it as we need.

2. Data Layer connection. Depending on the alternative we choose to work with, we'll proceed in one way or another. Thus, if we select App Engine's Datastore API we'll need to create the entities to hold the data on App Engine, and access them via Manager objects and SQL-like provided tools in the code lines of the main classes of our application. If instead of that we work with Data Source Library for Visualization API we'll also need to create the classes for the data on App Engine, but access to its URI trough the methods defined in the Visualization API.

   On the other hand, if we work with Google Spreadsheets, we just need to ensure of getting the correct URI of the needed files hosted in Google Docs, and work with it via Visualization API functions in the application's code.
   Finally, if we've choose to work with the Base Data API, we need to obtain a item feed URI to store our items and parse correctly the xml answers of our http GET queries to obtain the data in the expected format for being displayed.

3. Access to corporative data. If instead or also working with data on the cloud we desire to access to data located on the physical infrastructure of the company it will be highly recommended to use Secure Data Connector. Trough App Engine we'll send requests to access the data, and after validate that ones, the information will be returned to Google Apps, so we'll be able to manage it as we desire. Should be noted that the process will depend on the system and structure of the local company databases.

4. Visualization Layer. Now we're going to take a look on how can we display data as charts. For this, we can select one of the two options given by Google Chart Tool. If we use the Chart API we just need to use App Engine to set the correct paths for the URIs that will contain the charts. On the other hand, using Visualization API, we need to load the corresponding libraries in the application's code and after that, proceed to work using the API methods.

5. (and 6.) User Access control. In order to ensure that the system is accessible just for valid users we'll use the Google Accounts API. To integrate it with App Engine, we just need to load its libraries to be able to access to some methods for redirect users to Google's authentication service and bring them back to the application when validated.

7. User interaction. System users will use the application via browser, so after typing the URI of it and validate themselves using the accounts system described above, the application will be accessible. Thus, they'll be able to navigate trough it, perform queries, obtain reports, and so on.

In conclusion, we can say that using the alternatives offered by Google Code to combine with Google Apps is possible to easily develop BI systems characterized by its sturdiness, fastness and flexibility.

As is said before, tools will be focused on the displaying of information in order to know the situation of the company, although the flexibility of the solution will allow us to add analysis, forecast and some other tools in case we need.

Also, we can retreat the fact of not working with a powerful database system indeed if we just focus on the data layer tools seen in the section 4.2 of this chapter. That may limit the scope of the solutions that could be developed.

However, we can see these solutions acting as a starting point for new and better ones. Due to the easy change adaptability that the systems build under the guidelines described in this chapter have, a change in the part referring to the data layer[18] or the addition of a new feature to the core would not cause any structural risk to the whole system.

Thus, replacing the alternatives offered by Google by a powerful database system composed of various sources with an ETL process to a data warehouse, adding some data marts and being accessible via SDC, we would increase the size of our system. Moreover, the security and scope of the solution will be improved. In that way, we could give the user the possibility to perform queries involving a large volume of data and also avoid that those ones could occasionally collapse the system.

All this possible improvements would allow us to focus our solutions in bigger companies.

That's why we can conclude by saying that Google Apps in combination with Google Code tools is an interesting platform for developers in order to build BI systems for corporations.

[18] We'll just need to ensure that the format of the data for display the visualizations still being the correct one.

5. The development of easyBIew

The aim of this chapter of the document is to exemplify the building of a BI solution based on the Google tools and libraries seen in the previous chapter. For this purpose, we'll see the complete development of a system named easyBIew. So, this chapter can be also seen as a kind of tutorial about how to create a Google-based BI tool for some corporation.

First, we will introduce the mentioned example tool explaining its aim, which will condition its aspect and characteristics.

Then, we are going to take a look to the design and technical decisions taken trough a brief explanation about the used technologies for reach the set goals. After that, we will see the different features of easyBIew, introducing all the options given to its users.

Finally, we will present a complete step by step guide about its building, where it will be possible to see an illustrated explanation of all development aspects. Furthermore, for concluding the chapter, some ideas for improving the original solution will be suggested and explained.

5.1. easyBIew

The name of the tool pretends to be a word-game between the IT discipline that involves it and a reflex of its main quote: the simplicity.

Through easyBIew we want to make easy what is not. It's searched to give to the users a friendly interface to allow them to know the situation of its enterprise and have, at the same time, knowledge from valuable information to take decisions in order to define the course of it.

The objective of the program is to be controlled by the user instead of control the user.

Fig.14 – Designed logo for the tool.

According to this principles, easyBIew will offer a light and friendly interface, where the needed information will be easily displayed. Instead of being a baroque application with lots of menus and options, users will find exactly what they need in the moment they need it.

5.2. Decisions taken

In this section we are going to see the decisions we took in order to build easyBIew. First of all, we will take a look to the technical ones. This will be done by explaining those selected alternatives from the ones offered by Google seen in the previous chapter. After that, we will see those decisions related with the design of the tool.

And finally, for concluding this section, we'll take a look also to other decisions related with the development of the system.

5.2.1. Technical decisions

In the following lines we're going to explain which alternatives of the ones offered Google we selected to develop our BI-system. Thus, we'll see trough a reasoned explanation the components that will conform easyBIew.

The first dilemma we have is on the choosing of the App Engine's development version. As is said, we've the alternatives of work using Java or do it with Python. In our case, we decided to use Java, so let's going to see now the reasons that make us take that choice.

Python is an interpreted and dynamically typed language used by lots of applications. Also, as is multi-paradigm allows different styles of programming: object-oriented, structured and functional. Moreover is a multi-platform language.

On the other hand, Java is a compiled and statically typed language, purely object-oriented and also used by lots of applications. Moreover has lots of accessible APIs and, as Python, is also multi-platform.

Thus, after seeing this brief description about them, let's focus on its differences.

Even Python is more concise, and that may make seen it as cleaner, Java is a language more structured and formal, so implementations using it minimize the potential errors and don't tend to chaos. Furthermore, these characteristics make it easier to re-use code to improve the original solution and also create applications for other devices.

Another reason to select Java is that usually, compiled languages are faster than interpreted ones.

So, we based the election of Java as a platform for develop easyBIew on these characteristics: the formality, its re-usable characteristic and its fastness.

Now, for continuing the explanation about the technical decisions we'll talk about the data layer. From all the alternatives exposed for it in the previous chapter, we are going to use the Spreadsheets one. As is fully compatible with Visualization API the process of gathering the desired data in the expected

format becomes easier and faster. Also, as is being part of the suite Google Apps, we will work with all the solution integrated, so will be very easy to set privileges to edit spreadsheet files, update that files with new information, and reflect changes in the monitoring visualizations appearing in the application.

Moreover, using Spreadsheets as a data store for our system we'll hardly reach the quote limits of App Engine, so we'll be also reducing costs for the corporation.

Against its using we can say that data is on the cloud, so that can create uncertainness to the company. A possible solution for this problem is to have replicated the Spreadsheets data in local data store systems. Another drawback is that the scope of the tool may be a bit limited. Spreadsheets is not the best alternative to have a big and powerful data layer and perform queries involving large volumes of data. However, as the concrete implementation of easyBIew is not thought to work with lots of data we'll avoid that problem.

Finally, for concluding this section, we'll expose the reasons of our choice for the visualization layer.

Even choosing Visualization API we lose fastness than if we work with Chart API, we earn the possibility of work with stored data in a comfortable way. Also, that data can be managed through the SQL-like interface, so we also earn freedom gathering the data we need as we need. And these are the main points that make us select Google Visualization API instead of Chart API as a tool for displaying data in our BI-system.

5.2.2. Design decisions

At this point, we are going to talk about the decisions taken in order to define how the application will be. For this, we will expose some requirements based on the principles of the tool seen in the previous section.

We should note that due to the characteristics of a system developed under the guidelines described in the chapter four, the application will be focused on displaying the concrete information that the user needs. Because of that we'll offer a dashboard panel with static charts monitoring some metrics about the company performance, and a query and reporting tool that will allow users to search for concrete information.

Thus, with this and the principles in mind, we can say that easyBIew will have a light interface. Excepting the dashboard panel, users will be able to see just one chart per page to not make heavy the understanding of it. Each time they decide to change the view topic or navigate and vary the data, will be possible to do it trough an easy drop down menu located at side of the chart.

Moreover, a logged user will have access to every part of the application, regardless of its current location on it. Just will be necessary being authenticated correctly.

Finally, we can say that because of its architecture, defined by the exclusive using of Google Code tools, easyBIew will be oriented to small and medium corporations.

5.2.3. Development decisions

Finally, for concluding the decisions section, we will take a look to those related with how will be the environment for work on the system development.

There are two possibilities of working with App Engine's Java version. The first one is download the SDK, configure it by hand creating the directories structure, copying the libraries, etc., and test the application using a web-application server such Apache Ant. The second one consists on download a plugin for Eclipse which includes the SDK already configured and prepared for start implementing, a local applications server to test it, and also an integrated application to upload it to Google's infrastructure.

So, due to its comfortableness we've chosen to develop the application with Eclipse and its complement for App Engine instead of doing it by hand and testing with Apache Ant.

Also, as we're planning to build a web-application, it will be interesting to use some tool in order to design its interface. For this, we're going to use Macromedia Dreamweaver. In that way we'll create some html and css templates to use inside our application's code.

Finally, to design and manipulate images we've choose to work with Paint.NET. As is a free tool that offers us an Adobe Photoshop-like interface full of features, that makes it a really interesting alternative to work with.

At this point we've seen all kind of decisions that will condition the building of easyBIew. The following sections, so, will be based on these guidelines.

5.3. Software features

In this part of the present chapter we are going to take a look to the different functionalities that easyBIew will offer to its users. As we've seen in the previous section, due to the characteristics of a system developed by Google tools, we are going to focus on the displaying of concise information about the situation of the corporation, in order to manage it appropriately and help also in the decisionmaking process.

Thus, easyBIew will offer to its users a dashboard panel and a query&reporting tool.

Dashboard

Through it users will be able to monitor some indicators about the company performance. The information will be displayed as charts, so the understanding of it will be really easy and comfortable.

Query&Reporting

This tool will allow users to easily search concrete information about its company in order to help taking decisions to define the course of it. Also, by using it, they will be able to focus on different topics, and filter its content as they select through an easy form menu located on the left side of the screen.

5.4. The development: step by step

Once we've seen the principles of the tool and an explanation about the decisions taken in order to build it, we are going to see a guide about its development. To make it follow-up easier we will go step by step, illustrating each point.

Thus, the first step will be the preparation of the environment for working. For this, as we're going to work with the Java version of App Engine we've downloaded the last versions of Java Runtime Environment and JDK, and set the paths correctly[19].

As we said, due to its comfortableness we've chosen to develop the application with Eclipse and its complement for App Engine instead of doing it by hand and testing with an application server such as Apache. So, the following thing we've to do is download the App Engine's complement for Eclipse[20].

That can be done trough the option "Install new software" from the Help menu of the program. Then, we should press "Add" button, enter the following URI in the field "Location" and press OK: *http://dl.google.com/eclipse/plugin/3.5*.

As a result we will get the possibility of install the Google Plugin and the Java SDK for App Engine as the following image shows:

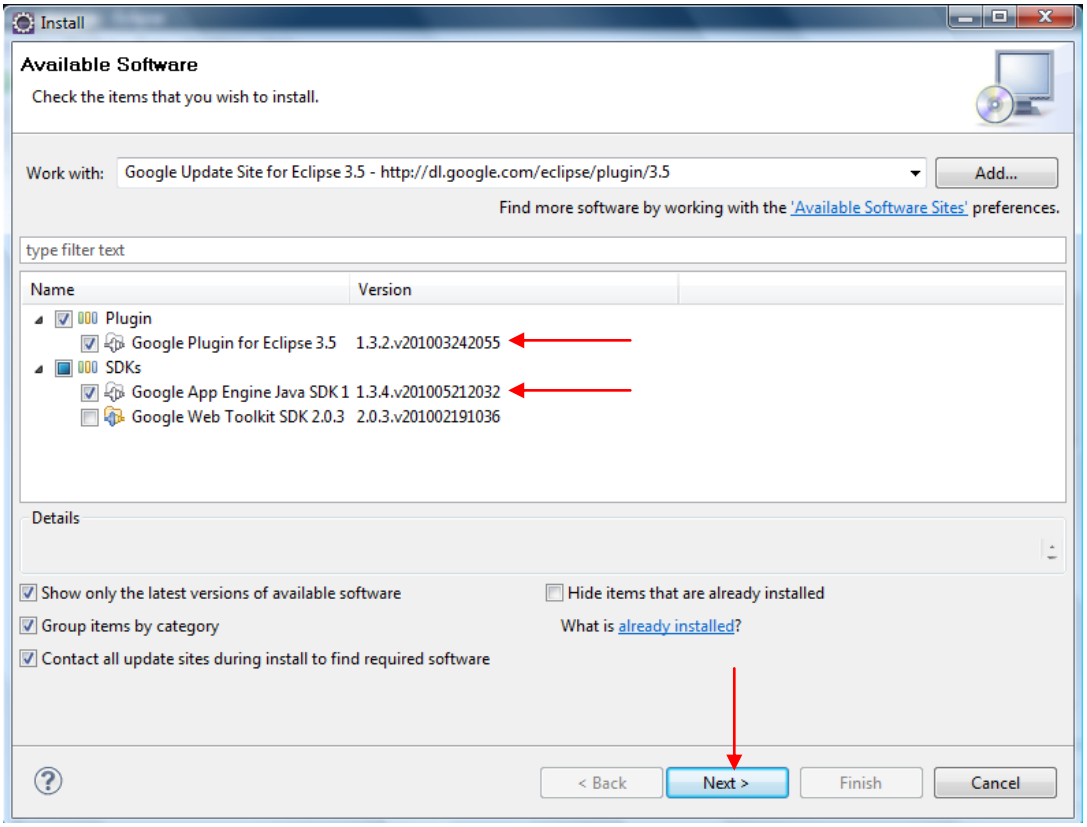


Fig.15 – Installing the Google Plugin and App Engine's Java SDK.

[19] The explanation assumes the knowing of perform this step.
[20] All the process described in this chapter is using the version 3.5 Galileo of the program.

Then, we should just click on the "Next" button and in a few moments we'll have the environment ready for start developing the application.

The following step we need to take is to activate the App Engine service in the Google account we'll use as an administrator or owner of the tool. For this, we must go to the following URI: http://appengine.google.com and log in with our account. Then, after clicking "Create an Application", it will be required to verify the App Engine's account via SMS. We'll receive an account code and we just need to enter it. As a result we will be redirected to a page where we'll create the application and define some of its settings.

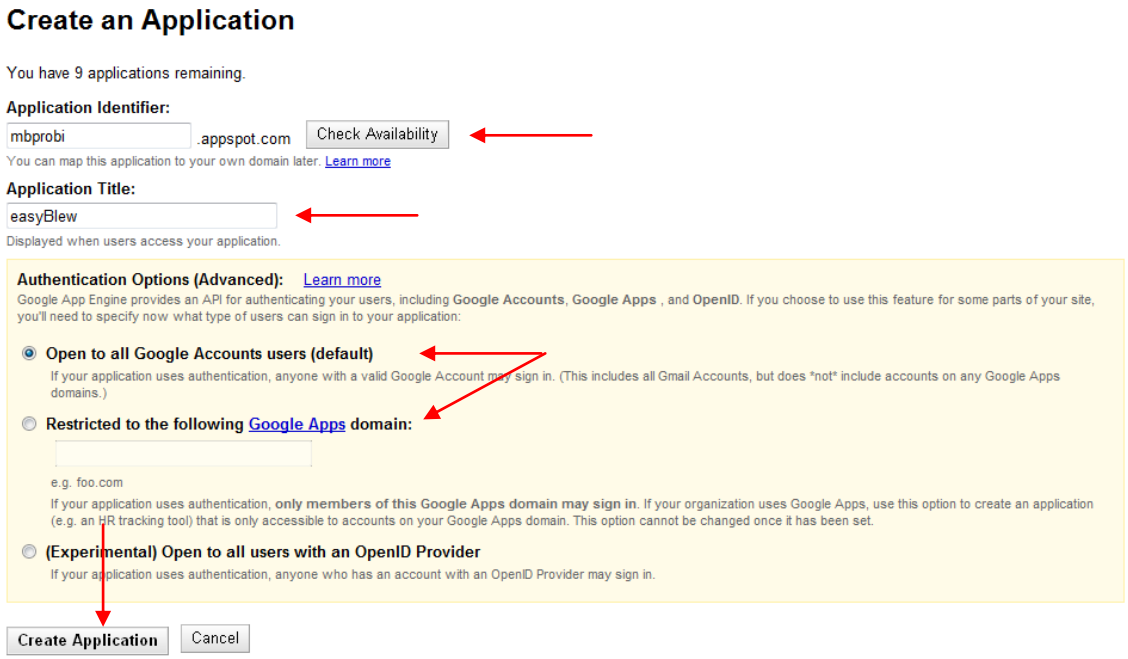


Fig.16 – Registering the application for App Engine on Google Apps.

As we can see in the image, we've chosen *"easyBIew"* as title and *"mbprobi"* as identifier for the application. So, we'll be able to access to the tool by typing *http://mbprobi.appspot.com* as a URI in our browsers.

Moreover, we can define the authentication mode of the system. Even it's possible to restrict it to a Google Apps domain, as we don't have any created to use, we'll choose to authenticate users just with Google accounts.

After clicking "Create Application" option we will be able to access to the App Engine's control panel for our application in order to see some statistics about its usage and also manage some aspects of it, such as permissions setting, control different versions and our quote limits, re-define its domain and so on. That feature will be useful later.

Once we've prepared the environment and defined the application settings, we can start developing it. For this, let's take a look first to the structure of a project

created using the Java version of App Engine's SDK. This one is based on WAR format for Java web-based applications.

```
easyBIew/
  src/
    ...Java source code...
    META-INF/
      ...other configuration...
  war/
    ...images, static Html files, data files...
    WEB-INF/
      ...app configuration...
      lib/
        ...JARs for libraries...
      classes/
        ...compiled classes...
```

Thanks to the using of the Google Plugin for Eclipse is not necessary to create these directories, the program will create them for us. So to start building our system we just need to select the option "Web Application Project" of the "File > New" menu of Eclipse, and define the names for the project and the package directory which will contain it. In the example we've chosen "easybiew" as a name for both.

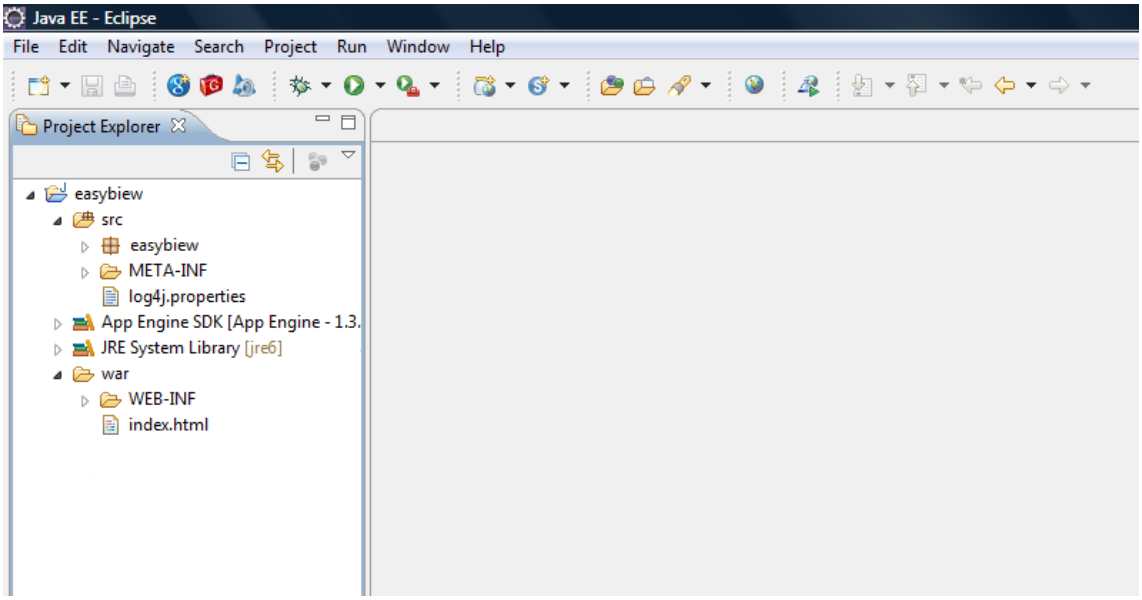


Fig.17 – Java WAR format directories structure created on Eclipse.

As a result, we'll have the environment with the structure of our project defined and having the necessary libraries to start working with it. So the following step will consist on start adding content to the easybiew package.

In order to build a user-friendly tool, the first thing we will do is create an interface and define its style using an html editor. As is said in the section 5.2, we've chosen Macromedia Dreamweaver for it.

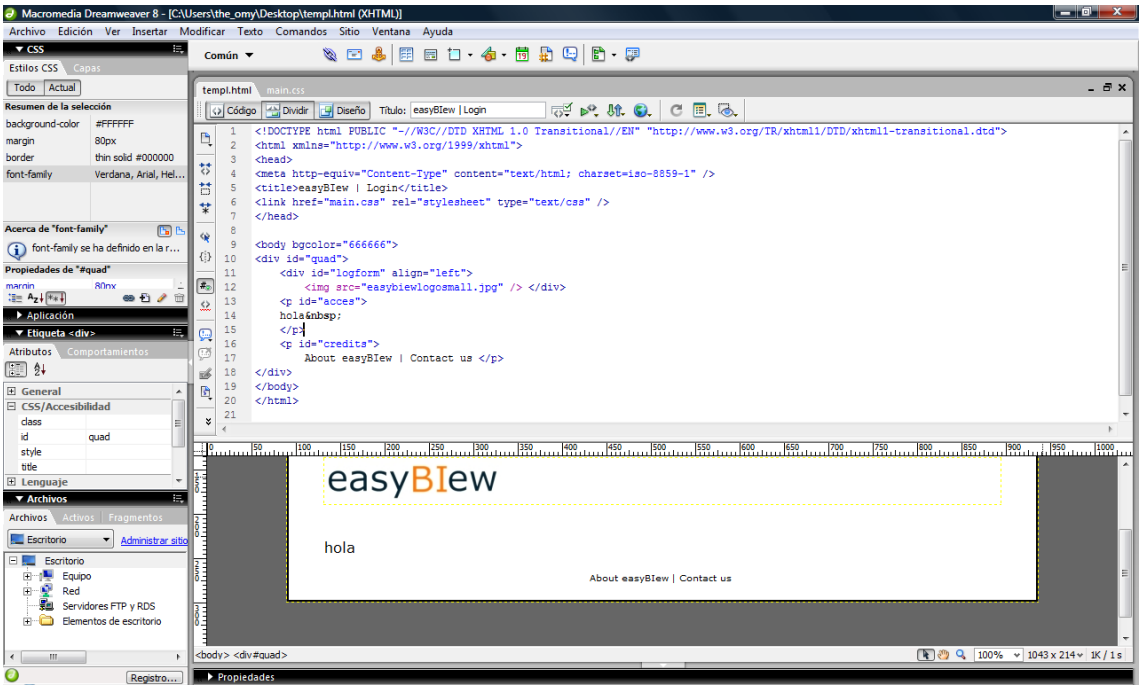


Fig.18 – Designing the html and css interface on Dreamweaver.

That interface will be used as a pattern for our system visualization. To use it we just need to move the css files we create to the war folder of our easybiew package directory previously defined in Eclipse, and copy the html code to use and adapt it on the system files we're going to write in the following steps.

To make easier the understanding of the easyBIew's architecture, we'll represent it on a UX Model's[21] navigational map-based diagram (Fig.19). So watching it the following content will be clearer.

[21] User Experience Model, is a RUP-based (Rational Unified Process) technique for design web applications. Describes how the dynamic content will be structured and organized in different screens, and how the user will navigate among those screens to execute the application use cases.[15]

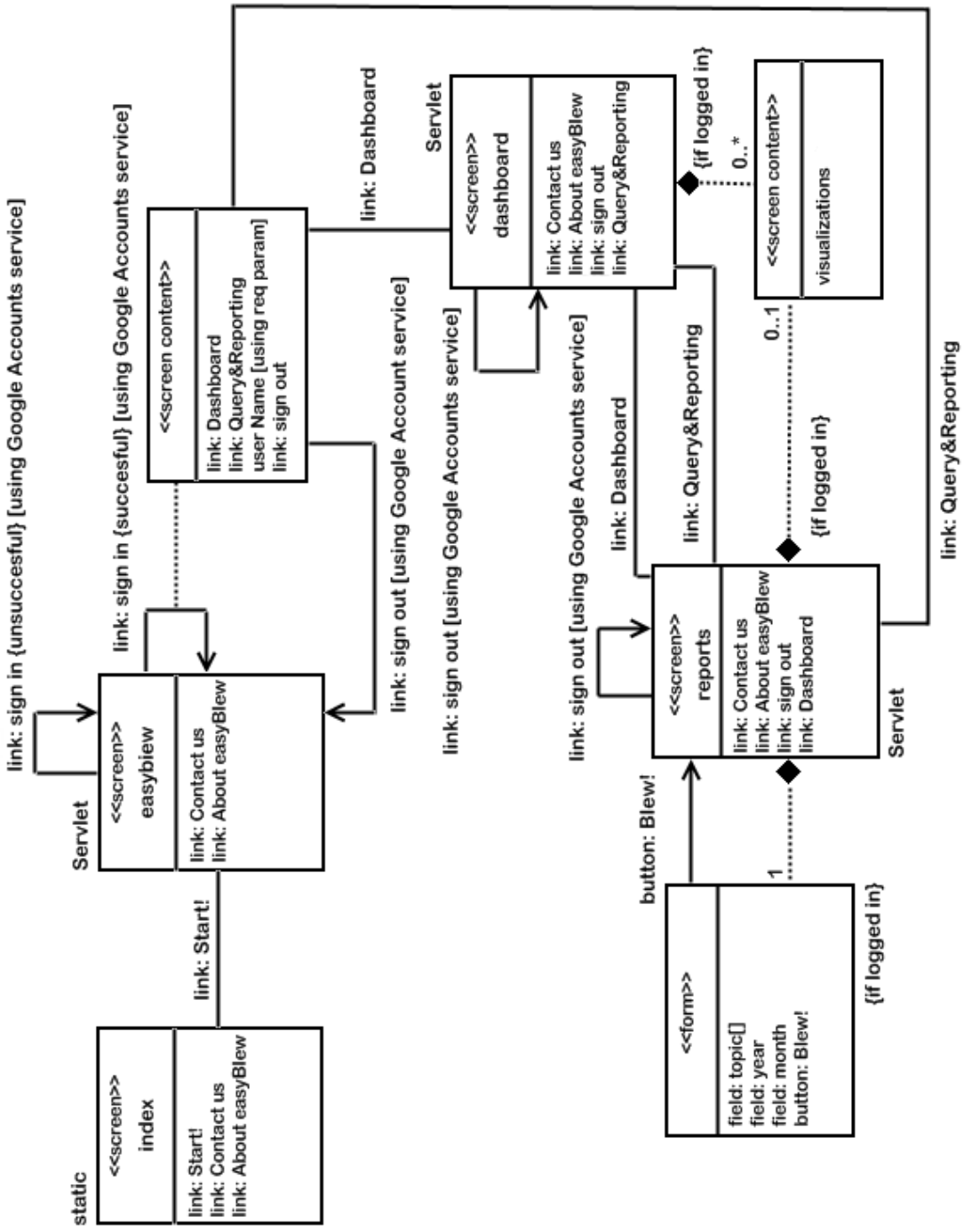


Fig.19 – easyBIew architecture.

On the next lines we will describe the building of each part of the previous diagram. First of all, we are going to talk about the static files. As these ones are based on simple html code, having always the same content, we're not going to extend so much the explanation about them and will assume that have already been designed using the html editor.

So, in order to include that files in our system, we just need to place the cursor over the package name and press the secondary button to display the options menu. Then, we should select the "New > File" option. Once we get opened the new file pop-up window, we should select the parent folder, easybiew/war in our case, and browse our file system in order to import the desired file. The following image shows that process.

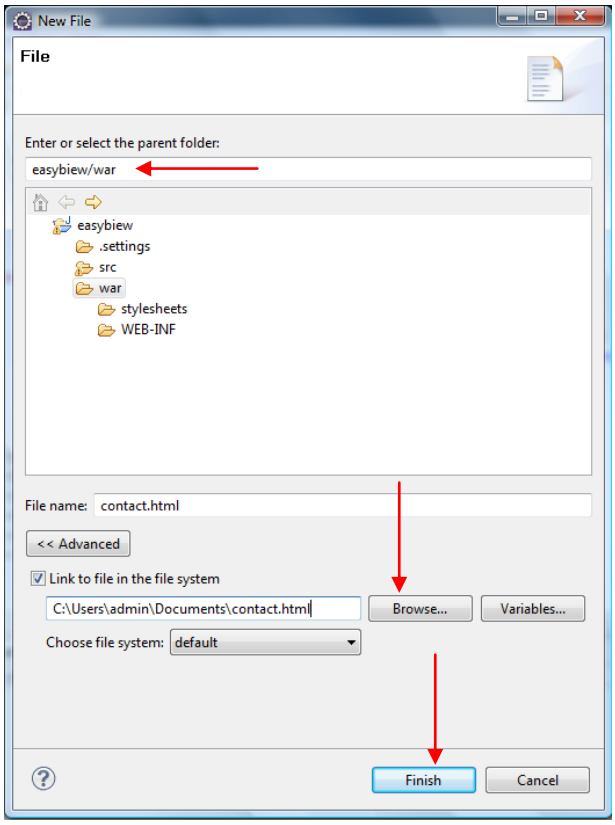


Fig.20 – Importing html files.

In the same way, we will add all the static files one by one. Just should be noted that we have also created a folder named "stylesheets" in order to hold the css files which will define the aspect of our interface. So, the code must take care about setting the correct path of these files in order to use them.

Moreover, as a convention, each page of the interface (static and dynamic ones) will have links to the Contact us and About easyBIew pages situated at its bottom.

Once we've seen how create the static files of easyBIew, the next step we will follow is the building of the dynamical parts of it. Applications developed using the

Java version for App Engine use the Java Servlet standard to interact with the web-server environment. Thus, we'll use that technology for the rest of our BI-system.

In order to start the explanation we'll talk about the easyBIew menu one. First of all it will be necessary to create the Java file which will contain the Servlet code. For this, we just have to repeat the procedure we did before to import the static html files, and instead of browse and select some file we just need to create a java one and include it in the easybiew/src/easybiew directory of our application.

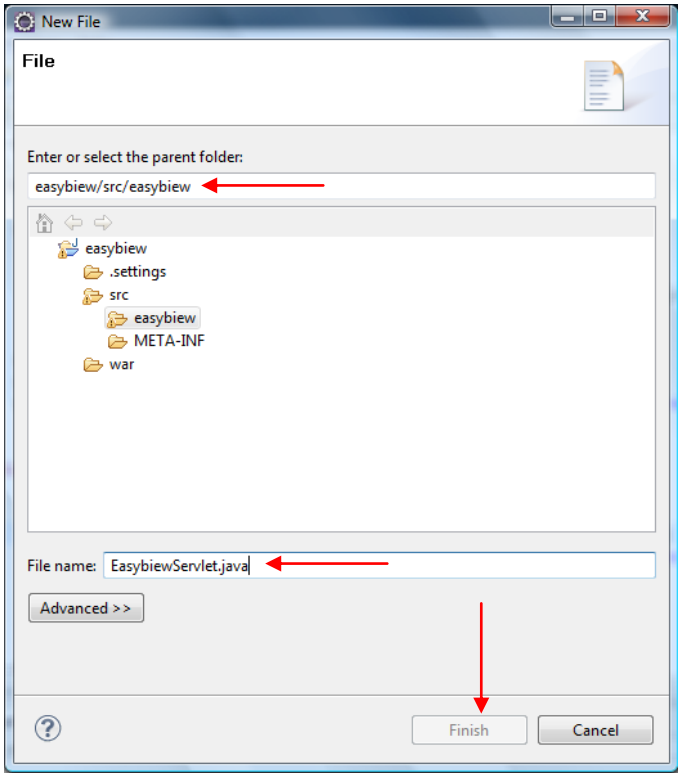


Fig.21 – Creating EasybiewServlet.java file.

Once we've the Servlet created we need to map it in the web.xml file situated in the war/WEB-INF directory of our application. So, we're going to include the following lines:

```
<servlet>
        <servlet-name>Easybiew</servlet-name>
        <servlet-class>easybiew.EasybiewServlet</servlet-class>
</servlet>
<servlet-mapping>
        <servlet-name>Easybiew</servlet-name>
        <url-pattern>/easybiew</url-pattern>
</servlet-mapping>
```

At this moment we have the Servlet file ready to be filled with the desired content. As the aim of this page is to offer the alternatives of working that an authenticated

users has, we need to ensure that the current user is a valid-one. For this, we'll use the Google Accounts API by writing the pattern lines below in our Servlet code:

```
//loading the API

import com.google.appengine.api.users.User;
import com.google.appengine.api.users.UserService;
import com.google.appengine.api.users.UserServiceFactory;

UserService userService = UserServiceFactory.getUserService();
String thisURL = req.getRequestURI();


if (req.getUserPrincipal() != null) {
        //display the menu options
}
else{
        //offer the user the possibility of log in
}
```

Watching the code above, we can see that after loading the API, we proceed to invocate the user service and save the current URI. Then, it will be necessary to include a check in the code to verify if the user is already authenticated and valid or not for access the application. That will be done by checking if the result of getUserPrincipal() method of the HttpServletRequest parameter is empty or not.

In case it isn't we'll proceed to display the content of the menu. Conversely, if not logged in, we'll show him the possibility of do it instead of the menu, by using the method createLoginURL() of the userService object previously created and passing in the current URI as a parameter.

```
<a href="" +  userService.createLoginURL(thisURL) + "">sign in</a> using your Google account.
```

So, in this way, the first time the user accesses to the page we'll detect that is not logged in. After authenticate himself will be redirected to the previously saved URI of the page, when this time, the getUserPrincipal() result will be already set in the request parameter and, in consequence, detected as a valid user. So, the menu options will be displayed.

The only thing we need to do now is provide an interface for the page. As we want to output it as html, we will need to use the out.println() method, so we must import the Java io library. Each html and Javascript code line of it will be printed using that function in our Servlet in the way that the following example shows:

```
import java.io.*;
import java.io.IOException;
…

resp.setContentType("text/html");
PrintWriter out=resp.getWriter();

…
out.println("var data = response.getDataTable();");
…
out.println("<div id=\"pie_chart\"></div>");[22]
…
out.close();
```

Thus, we're going to use and adapt the template we've created in the html editor before, according to what the menu page should contain. As is said, we just need to put each line inside the print function, and set the path of the corresponding css file. In our case, we should point to stylesheets/main.css.

So, in order to offer to logged users the possibility of choose the tool's options we will include links to the dashboard panel and the query&reporting Servlet pages. Moreover, using Google Accounts API we're going to display a greeting to the user and the possibility of logout. That will be done trough the getUserPrincipal().getName() method of the request parameter and createLogoutURL() one of the userService object, respectivelly.

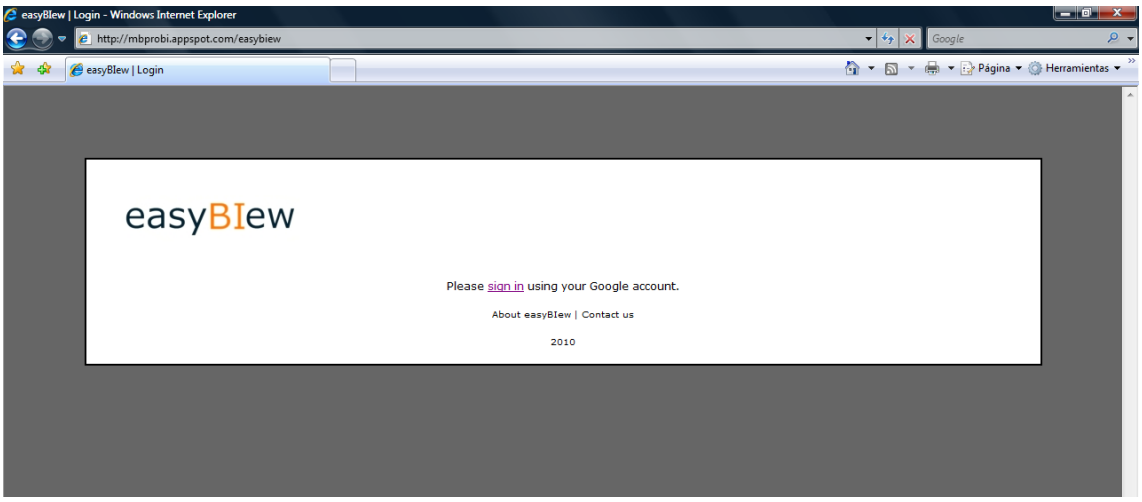


Fig.22 – Aspect of the page when the user is not already logged in.

[22] Each " symbol of our code should be replaced by \" inside out.println() method, if not, the compiler will detect it as an error in the String parameter.

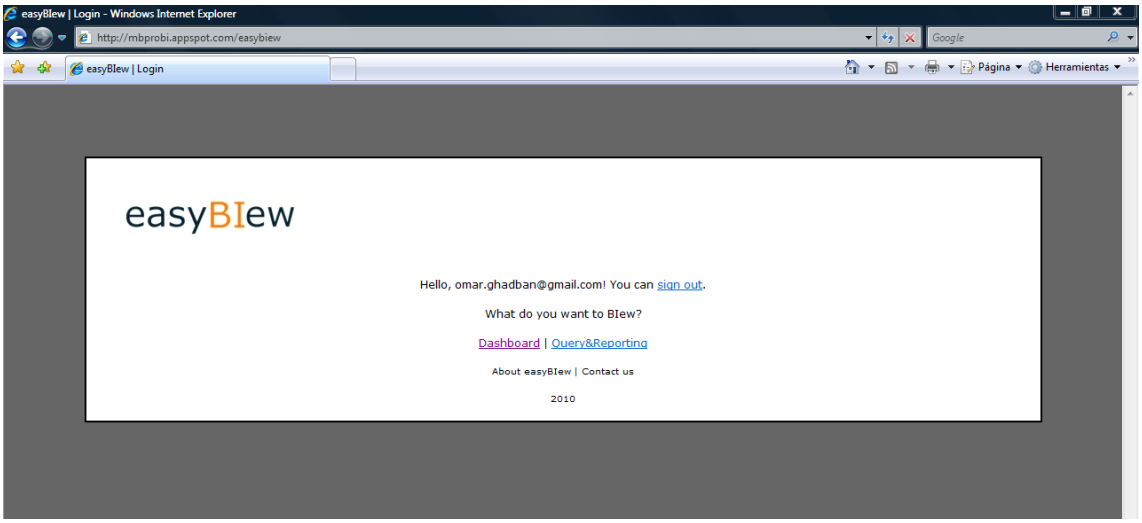


Fig.23 – Aspect of the page when the user is logged in.

Once we've seen the implementation of the menu page we will continue the explanation of the dynamical parts of easyBIew by describing the dashoboard panel.

The procedure about creating and mapping the Servlet will be exactly the same we've done before for the menu, so we'll focus the explanation on the design of the page.

Thus, first of all we should invocate again the Google Accounts API in order to verify if the user who is accessing to the page is a valid one or not. For this, we'll also proceed like we did before: in case is already logged and a valid user the dashboard panel will be shown to him, and if not, we'll offer the possibility of login.

As the aim of that section is to monitor some metrics about the company performance in order to know easily the current situation of it, we'll display some charts, based on the data located in a Spreadsheets file, which will act as indicators. So, to do that, we'll need to use the Visualization API.

The first step will be the importing of the libraries inside our html code. First of all we'll load the AJAX API, and then the visualization library with the concrete visualization package we need.

```
out.println("<script type=\"text/javascript\" src=\"http://www.google.com/jsapi\"></script>");
out.println("<script type=\"text/javascript\" >");
out.println("google.load(\"visualization\", \"1\", {packages:[\"corechart\"]}); ");
```
[23]

Once we've loaded the libraries, we should invocate the methods of the API to retrieve data from the Spreadsheets file where is located. So, before continuing the explanation, we are going to show how is possible to obtain its URI.

[23] Should be noted that in the example we'll use the pie and the line chart, so according to the documentation of this concrete visualizations, we've to load the corechart package.

First of all, we need to create a spreadsheet file and populate it with the data we want to display. After that, we need to set the viewing permission correctly. For that, we will go to the Google Docs items display page, select the document and access to the Sharing settings menu of the Sharing option.

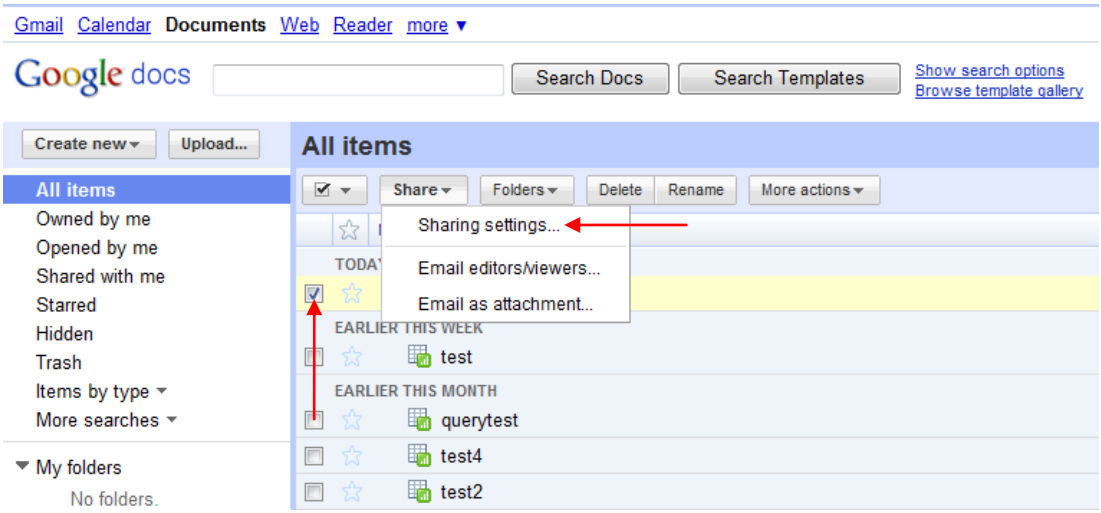


Fig.24 – Sharing a Spreadheets file on Google docs menu.

After clicking that option a pop-up window menu will appear. It will be interesting to restrict the permission of viewing to a concrete Google Apps domain name, but as is said before we don't have any created, so instead of doing that we'll set the permission as allowed to everybody who has the link. For this, we should click on the Change option, and select "Anyone with the link". After that, we should press "Share" button and the file will be accessible to Visualization API.

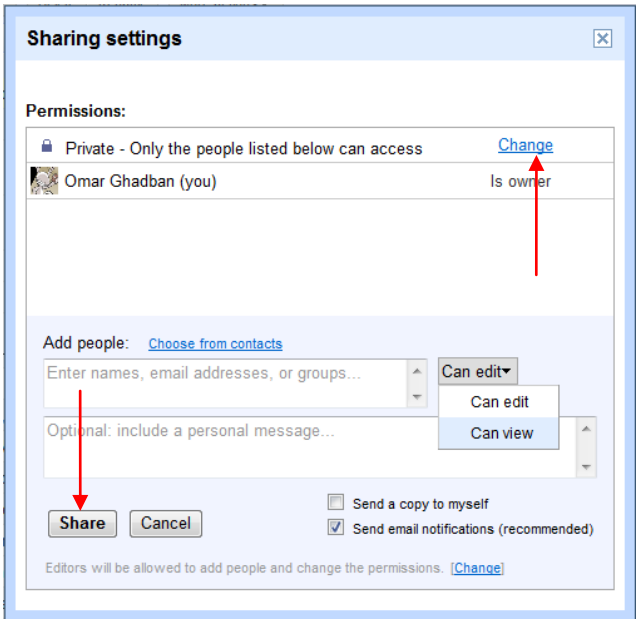


Fig.25 – Setting the viewing privileges of the selected file.

Once we've set the viewing permission we need to obtain the correct URI path for our data. So, we'll open again the file and select the Gadget option of the Insert menu. After selecting a Gadget corresponding to interactive charts tool (no matter which) a pop-up window will be opened.

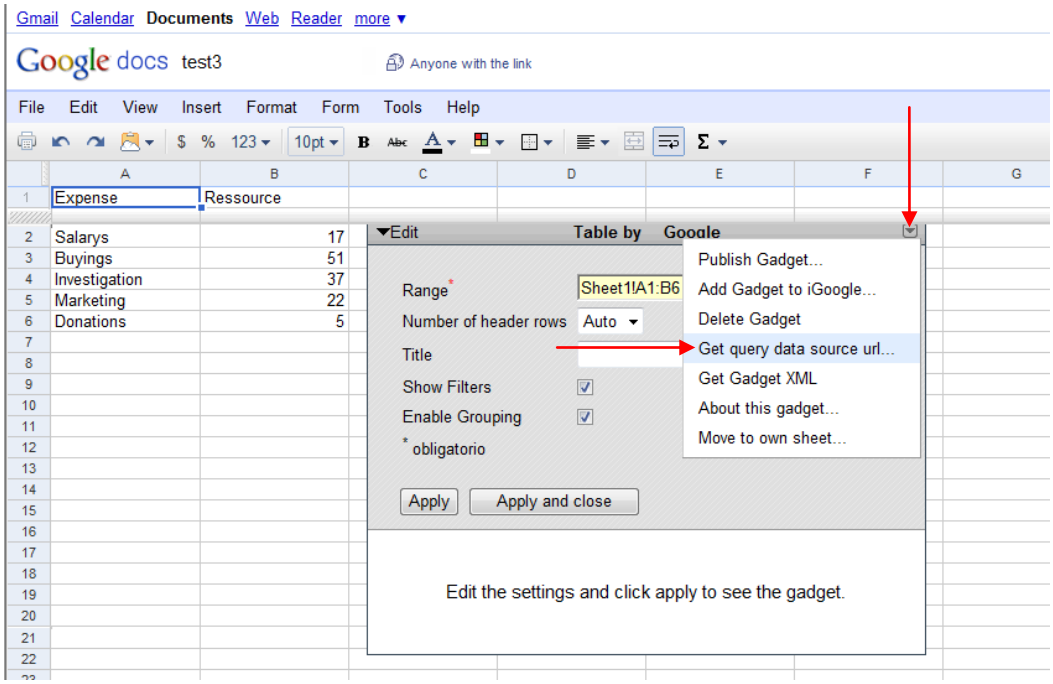


Fig.26 – Obtaining the file's URI to build our data source's URI.

As is illustrated on the image above we need to click on the top-right corner and select the option "Get query data source url". Then, the URI address of the file will be shown to us. Sometimes we don't get the exact URI for our data, so we must ensure of selecting the correct range of values, and the correct sheet. In our case, we'll select A1:B6 and 0 respectively. Thus, the URI we'll use in the Visualization API query will look as the following:

http://spreadsheets.google.com/tq?key=0Ahc6rLjB_skwdDVxdXBqeDVjQWFkWWIxWThUR3F2QlE&range=A1:B6&gid=0&headers=-1

After that break to explain how proceed with Spreadsheets to obtain the data URI, let's continuing the explanation about the building of the dashboard panel. So, the following step will consist on calling the methods to gather that data and display it. For this, we'll proceed by calling a callback function to initialize the charts we want to visualize.

```
out.println("google.setOnLoadCallback(initialize);");
out.println("function initialize() {");

out.println("var query_pie = new
google.visualization.Query('http://spreadsheets.google.com/tq?key=0Ahc6rLjB_skwdDVxdXBqeDVjQWFk
WWIxWThUR3F2QlE&range=A1:B6&gid=0&headers=-1');");

out.println("query_pie.send(handleQueryPieResponse);");
out.println("}");
```

As we can see in the code lines above, we are calling the method google.visualization.Query() passing in the URI of the file as a parameter. After that, we should send the result to a handler function, which will be the one we'll use to draw it.

```
out.println("function handleQueryPieResponse(response) {");
out.println("if (response.isError()) { alert('Error in query: ' + response.getMessage() + ' ' +
response.getDetailedMessage()); return;}");
out.println("var data = response.getDataTable();");
out.println("var chart = new google.visualization.PieChart(document.getElementById('chart_div'));");
out.println("chart.draw(data, {width: 400, height: 300, title: 'Company Expenses'});");
out.println("}");
```

The handler function needs to check if the response of the query to our data source is correct or there are some errors. That can be done by looking the response.getParameter() result. After being sure that there were no errors, we must obtain the data table trough the method response.getDataTable() and create the visualization. As we want to display the data as a pie chart we should invocate the google.visualization.PieChart() function passing in the div tag that will contain the chart as a parameter. Finally, we just need to call the draw method of the chart passing in the data and some display properties.

That procedure will be exactly the same for each chart we want to display in the dashboard panel. So now, to finish the building of that section we just need to use the html template created before in the same way we did on the menu page and perform also the check to see if the current user is authenticated or not.

Moreover, in order to respect the requirements previously defined we'll add links to the contact us and about easyBIew sections, and also to the query&reporting page we'll build next.

Finally, as a result, every authenticated user will be able to see the dashboard panel containing some charts in order to monitor the situation of its corporation (Fig.27).

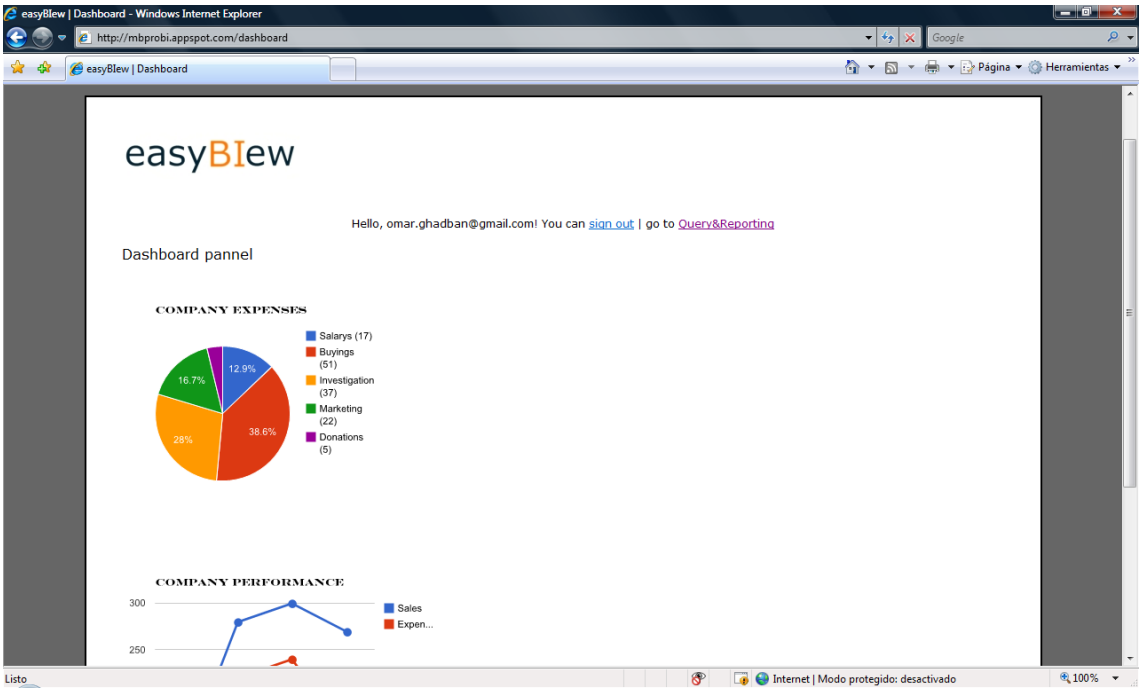


Fig.27 – Aspect of the dashboard panel for a logged user.

Once we've seen how to implement the dashboard section, we are going to take a look to the development of the query&reporting one. As the aim of that section is to process user queries in order to show exactly what they want, we'll need to display the charts depending on what they select trough a form menu.

Thus, instead of work just with the method doGet() of the Servlet we are going to use also doPost(). The first one will check if the current user is an authenticated one in the same way we did before and also, in affirmative case, will show the form to select which information is going to be displayed. The second one will gather the parameters passed in via form and build the chart to display based on the user request.

In order to respect the requirements previously defined we're going to show just one chart per query to not overcharge the screen and make heavy the understanding of the information.

Also, should be noted that to exemplify the performance of this part of the tool we are going to work with a 3-sheet Spreadsheets file as a data source for our visualizations. The first one will refer to data about company sales, the second to expenses and, finally, the third one to projects being developed.

For this, we've created a document and proceed in the same way we did before to set the view privileges and getting its URI. Instead of reference each sheet with the gid parameter of the URI we'll do it via sheet one with its given name. In this way, the address will look like:

http://spreadsheets.google.com/tq?key=0Ahc6rLjB_skwdHJaRTB2MG50RTRlVEFYTzdyTHJlS1E&range=XX:YY&headers=-1&sheet="sheet_name"

Moreover, we've decided to display the information contained in the document as tables, and allow users to filter it by year, month or both. To apply different filtering, obtain aggregated information, and so on, we just need to play with the query language offered by Visualization API.

So, all this considerations will make the reports Servlet implementation be based on the following pattern:

```
public class reportsServlet extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws IOException {

        …
        if (req.getUserPrincipal() != null) {
            //Display the html form with the fields that will allow the user to select and
            //filter the information.
            //Send it via method post to the same Servlet: reports
        }
        else{
            //offer to the user the possibility of log in
        }
        …

    public void doPost(HttpServletRequest req, HttpServletResponse resp)
    throws IOException {

        //gathering form's sent data
        String year= (String)req.getParameter("Year");
        String topic= (String)req.getParameter("Topic");
        String month= (String)req.getParameter("Month");

        …
        //loading the Visualization library we'll use to display the data
        out.println("google.load(\"visualization\", \"1\", {packages:[\"table\"]}); ");
        …

        //As sales and expenses have the same metric for data range we include both in the
        //same case.
        if(topic.equals("Sales")||topic.equals("Expenses")){

            //obtaining data from the Spreadsheets file.
            out.println("var query = new
            google.visualization.Query('http://spreadsheets.google.com/tq?key=0Ahc6rLj
            B_skwdHJaRTB2MG50RTRlVEFYTzdyTHJlS1E&range=A1:C42&headers=-
            1&sheet=" + topic + "');");

            //check if we're filtering per year, month or both
            if(!year.equals("")){
                if(!month.equals("")){
                    //setting the query via the SQL-like interface
                    out.println("query.setQuery('select B,C where year(A)=" +
                    year + " and month(A)=" + month + "');");
                }
                else{
                    out.println("query.setQuery('select month(A),B,C where
                    year(A)=" + year + "');");
                }
            }

            //Continue on the following page…
```

```
                    else{
                        if(!month.equals("")){
                            out.println("query.setQuery('select year(A),B,C where
                            month(A)=" + month + "');");
                        }
                        else{
                            out.println("query.setQuery('select
                            year(A),month(A),B,C');");
                        }
                    }
                else{
                    //if we are querying for projects we need to focus on a different range
                    out.println("var query = new
                    google.visualization.Query('http://spreadsheets.google.com/tq?key=0Ahc6rLj
                    B_skwdHJaRTB2MG50RTRlVEFYTzdyTHJlS1E&range=A1:F6&headers=-
                    1&sheet=" + topic + "');");

                    //do the filtering checkout in the same way above

                }

                //send the data to the handler function
                out.println("query.send(handleQueryResponse);");
                …

                //html code including again the form to allow users to filter the data or change it and
                //the div tag to display the chart
                …

        }
}
```

After seeing the directives that the code of our reports Servlet should follow, and for concluding the description about the building of that section, we'll take a look to some details we'll include to the visualizations in order to make the understanding of the information more easy and comfortable.
The first one is the addition of an explicative sentence about the information that we're displaying in a concrete moment. For this, we must include a check to the current topic parameter and its different applied filters to set the sentence appropriately.

```
if(topic.equals("Sales")||topic.equals("Expenses")){
        if(!year.equals("")){
                if(!month.equals("")){
                    out.println("<p id=\"seccio_content\">Displaying " + topic +
                    "&nbspof " + month + " , " + year + " (amount
                    expressed in Milion euro)</p>");
                }
                else{
                    out.println("<p id=\"seccio_content\">Displaying " + topic + "&nbspof
                    the year " + year + " (amount expressed in Milion euro)</p>");
                }
        }
        else{
                out.println("<p id=\"seccio_content\">Displaying the complete balance of " +
                topic + "&nbspof the company (amount expressed in Milion euro</p>");
        }
}

//Continue on the following page…
```

```
else{
        if(!year.equals("")){
                if(!month.equals("")){
                        out.println("<p id=\"seccio_content\">Displaying " + topic +
                        "&nbspstarted on " + month + " , " + year + "</p>");
                }
                else{
                        out.println("<p id=\"seccio_content\">Displaying " + topic +
                        "&nbspstarted on the year " + year + "</p>");
                }
        }
        else{
                out.println("<p id=\"seccio_content\">Displaying the complete list of " + topic +
                "&nbspof the company</p>");
        }
}
```

The second detail is based on the using of the formatter for table columns. Thus, in order to help the understanding of sales and expenses information we've decided to include a bar chart inside the column referring to the economic value amount. For this, we just need to create a bar formatter object by calling the method google.visualization.BarFormat() and apply it to the concrete column of our data before draw it.

```
//checking if we are displaying Sales or Expenses and not Projects
if(!topic.equals("Projects")){

        //creating the formatter, and defining a bar width of 80px
        out.println("var formatter = new google.visualization.BarFormat({width: 80});");

        //detecting where is the correct column to apply the format
         String col_form;
         if(!year.equals("") && !month.equals("")){
                col_form="0";
         }
         else if((!year.equals("") && month.equals(""))||(year.equals("") && !month.equals(""))){
                col_form="1";
         }
         else{
                col_form="2";
         }
         //applying the format to the column
         out.println("formatter.format(data," + col_form + ");");
}

out.println("chart.draw(data, {allowHtml: true});");
…
```

The last step we should follow is the including of links to the dashboard panel, contact us and about easyBIew pages in the same way we did before.
Thus, after applying these directives we'll have already a functional query&reporting section for our BI-system.

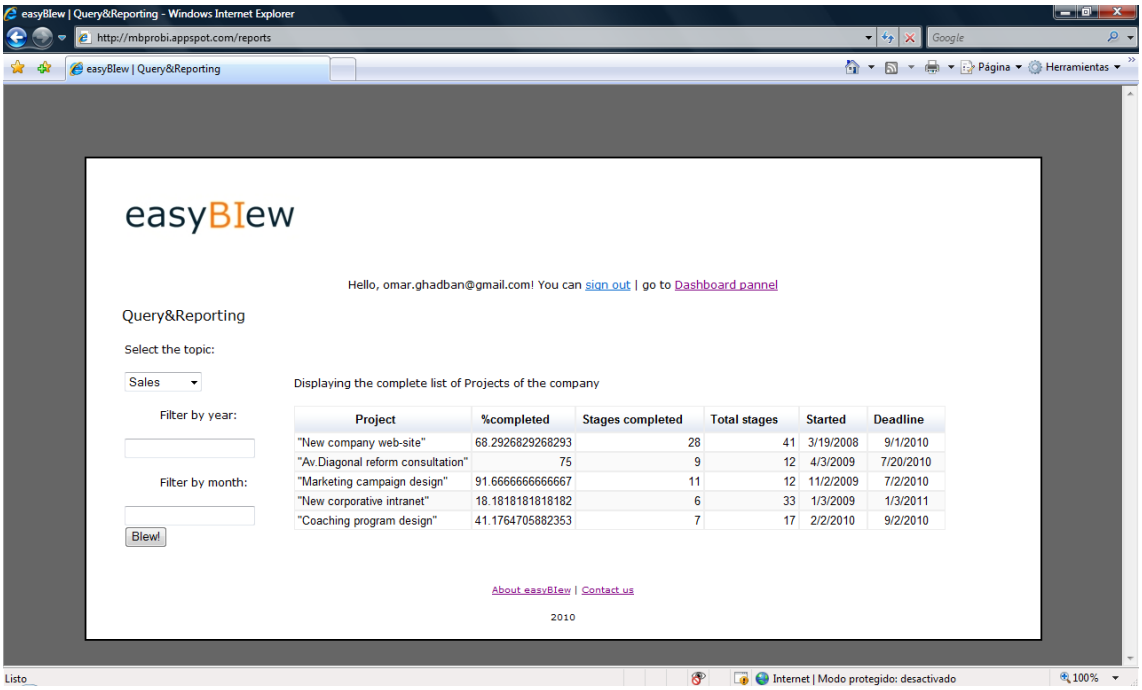


Fig.28 – Aspect of the Query&Reporting section displaying all the saved projects.

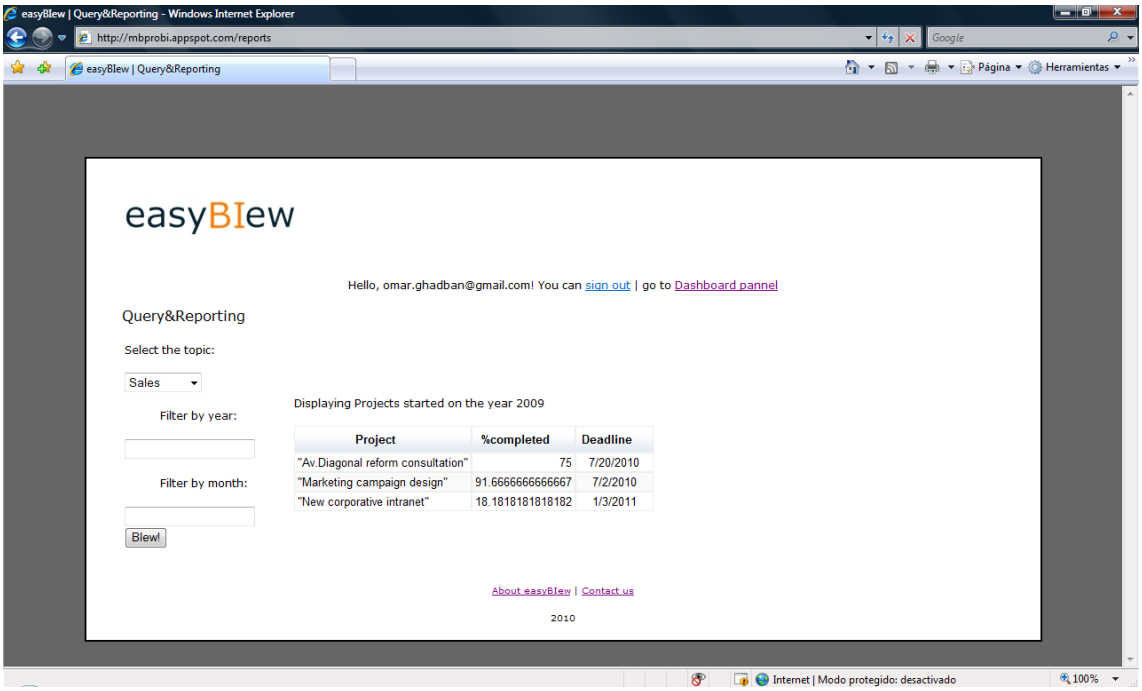


Fig.29 – Aspect of the query&reporting section after filtering results per year.

Once we've seen the complete development of easyBIew, let's concluding that section with a brief explanation about how is possible to test, run and upload the application using the Google Plugin for Eclipse.

Thanks to the internal web-server included by it, we can test each change we make to the program in our local machines before uploading it to Google's infrastructure.

For this, we just need to access to the "Debug As>Web Application" option of the Run menu. After compiling the code, the application will be running on the port

8888 of our local machine. So, we just need to type http://localhost:8888/ in our browser to view it.

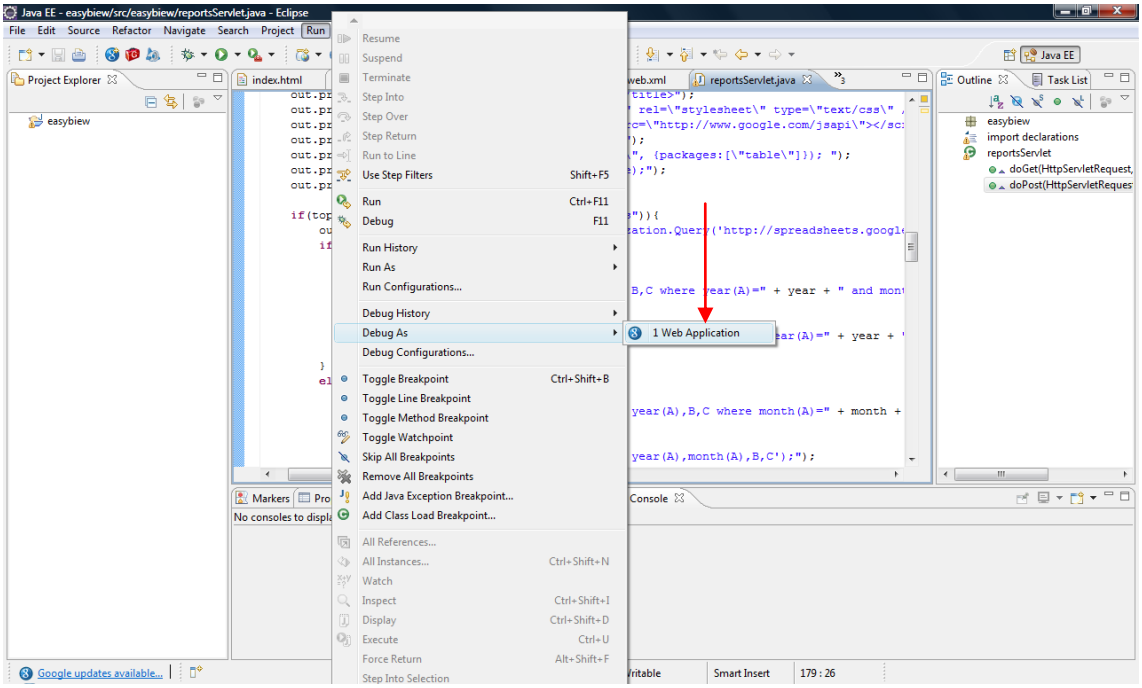


Fig.30 – Debugging easyBIew to test it on our local machines.

Otherwise, when we are ready to upload the application to make it accessible to its users, we will proceed by using the *AppCfg* feature of the Plugin. For this we just need to click on the App Engine button situated on the top menu of Eclipse, and introduce the Google account owner of the tool and its password on a pop-up window that will appear. After a few seconds we will be able to access the tool by typing its URI in the browser. In our case, http://mbprobi.appspot.com.

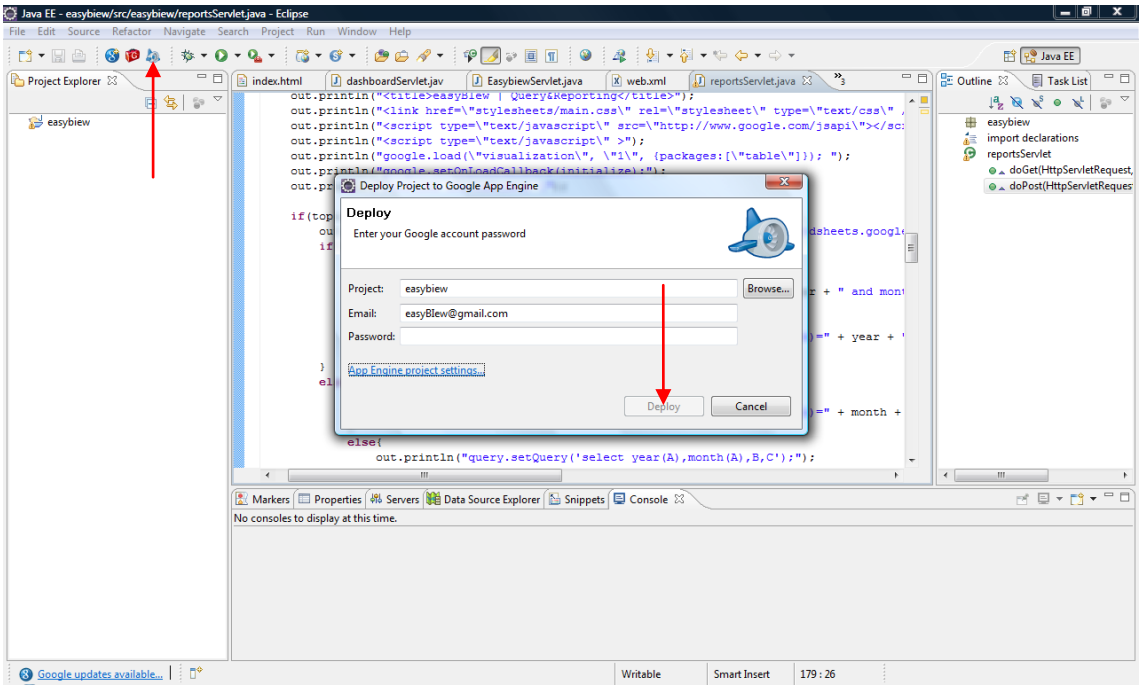


Fig.31 – Uploading easyBIew to App Engine through Eclipse.

So, as we've seen, we've created a completely functional BI tool easily and fast just using the alternatives given by Google.

5.5. Improving the solution

After seeing how we built an example BI web-application using some of the options explained in the fourth chapter of this document, we are going to propose some alternatives in order to improve the original solution.

As is said, systems developed under the given guidelines can be seen as a starting point in order to obtain new and better ones. So, in this chapter we will take a brief view to some functional and technological improvements that can be applied to easyBIew to perfect it.

5.5.1. Functional improvements

In this first section we will present some possible improves in order to add new features to the tool and make it more comfortable for users and administrators. For this, we are going to see and explain them one by one.

Customizable Dashboard

Instead of having a static dashboard panel with the visualizations added in the moment of building the tool, it will be really interesting to offer to the user the possibility of parameterize and select what they need to monitor exactly. The application should also save the user preferences to display the chosen charts each time the user access to its dashboard panel with its account without having to select them every time.

Using the Datastore API seen in the section 4.2.1 we can save the information about the indicators that user select, with its applied filters, and each time the user loads the page generate dynamically as much queries and container tags as visualizations saved by the user.

Administration console

In order to exploit the Google Accounts API, it will be interesting to add a section in the tool reserved for administrators. App Engine provides the necessary infrastructure to implement that kind of tools by offering the possibility of perform administrator logins.

This new feature will be useful to define some aspects about the performance of the application, such as apply changes to the way on how charts are displayed and manage the system used to store data, by including, updating or deleting some metrics over the company.

Thus on easyBIew, and after setting the correct permissions, the administration account will be able to access to Spreadheets files to add new information rows,

and update the Query&Reporting form menu directly from its console inside the application.

Forecasting tool

Another interesting feature to be added to easyBIew may be a tool to foresee future economic behaviors based on the analysis of the data stored by the company and the influence of some external variables introduced by the user. In that way, we would provide another strong weapon in order to face decisions to define the company's course.

Using of a Google Apps domain

Instead of doing the account validation just using Google Accounts and allowing the access to valid Google users, it would be interesting to restrict it to a concrete Google Apps domain. Thus, just users belonging to that domain would be able to access easyBIew. In that way, we'll exploit more the Accounts API features.

Mail Reports

With this new feature it will be possible to exploit another of the App Engine's characteristics, the using of cron jobs and the Mail sending API. Trough it we could offer to the users the possibility of receive periodically on their mail address reports about the indicators they choose. Once again, the using of another place to store user preferences, such as Datastore API, will be needed.

Thus, will be necessary to create a section in the application where users will select which information want to know, which filters apply to it, and define also how often receive it. In that way we'll start a process to create, map and activate a new cron job.

Also, we will need to create a hidden code file for the application (and specify it in the configuration file) which will be called by the cron job and will be also the responsible of performing the creation of the mail report by gathering its data regarding the concrete user preferences, and preparing and sending it using the Mail sending API.

Alerts tool

For concluding the functional improvements for easyBIew we'll purpose the addition of a tool for display alerts based in the Google AJAX API for feeds and cron jobs. With this, users would be able to configure and define in which information and metric is necessary to establish a check in order to launch or not an alert when happen. Otherwise, that alert can also be defined by the administrator if we add also the Administration console defined before.

So, for implement this we'll need again the using of a place to save user preferences. Trough a new section on the application, the user will select on which information topic focus, and the event that will activate the concrete alert. That checkpoint will be saved.

On parallel, a cron job will be the responsible of activate a hidden part of the tool that will perform the checking based on the user preferences. It will proceed gathering the necessary data from its source and parsing it to get the searched information to check. Each time it detects that the activation condition for some indicator acknowledges, will generate an rss feed containing information about the event. That feed will be saved.

Thus, every time that a user access to the tool with its account, we will show the corresponding alert feeds via AJAX API for feeds on real-time. In that way the system will become more proactive, and when a problem appears instead of search for it, it will come to us.

5.5.2. Technological improvements

Once we've seen the improvements related with the application functionality, we're going to take a look to some technical changes or additions that can be applied in order to obtain a better solution. As we did before, we will explain them one by one.

Increase AJAX usage

With this improvement we would earn a better user-friendly interface and an improved performance of the application. Visualization API uses AJAX indeed, so we can use one of its methods in order to refresh charts information every defined interval. For this, we just need to call Query.setRefreshInterval() passing in the refreshing time as a parameter.

Also, including AJAX on the Query&Reporting section we will be able to obtain the results of our queries without needing refresh the page, so the using of it would be really more comfortable for users.

Moreover, GWT may be helpful if we decide to increase the using of AJAX. For this, is also possible to work with a Plugin for Eclipse.

Redefine the Data Layer

Google Spreadheets offers us an easy and fast way to store data to display. But if we scarify these characteristics by changing the using of it for a powerful database system the scope of our solutions will increase. Thus, if instead of working with data on the cloud, we define a data layer composed by various local sources, an ETL process of them to a datawarehouse and the using of

some datamarts we would obtain a powerful solution able to process lots of data.

Use of XML

Another possible improvement that can be applied to our solution is to work with an XML database, as for example using Google Base Data API as is described in the chapter 4.2.3.

With it we would be able to exploit the advantages of the XML using, such as obtaining an application's data layer fully compatible with Java. Also, would be compatible to any application's core capable of processing XML (not depending on the platform), so we would increase portability. Our data layer would be totally independent of the system that uses it.

Moreover, as is an extendable language we would have freedom to create the tags which will define our data layer as we want, and re-use it when we consider.

Furthermore, the using of it will allow us to work with XSLT for the application's interface.

Finally, after have seen the possible improvements for easyBIew, we'll conclude this chapter presenting the conclusions extracted of its development.

As we've seen, we've build a totally functional BI-system using the guidelines described in the chapter 4.5 of the document and proving that is possible to do it in a fast and easy way. Also, by doing it, we have seen the independency between its different components, so all the changes or improvements suggested in the section above will not cause a critical impact to the whole system.

Even the possible technical changes applied to the data layer would carry the addition of some coding in order to gather the data and parse it in a different way, as we have developed a system with a structure completely adaptable to changes that process wouldn't be costly for the company.

All of this can make us say that Google Apps, in combination with Google Code APIs, is really a good platform to develop BI applications for corporations.

6. Conclusions

After seeing the studio about the different alternatives offered by Google to use in combination with Google Apps for develop BI-systems, and the step by step development of an example tool named easyBIew, we are going the expose the conclusions extracted by doing that project.

First of all, we've seen that the decisionmaking is something common in a company's day by day in order to define the course of it. But decisions should be based in some objective information. With this, we'll reduce our uncertainty and we'll be allowed to take better decisions which can provide us a competitive advantage.

Operational systems of the corporations generate and consume lots of data, and we need something to gather and transform it into information to, later, extract knowledge. BI discipline provides us that connecting point between business and the company management.

Thus, enterprises with a BI system correctly aligned with the business strategy, will be able to manage themselves and anticipate to future events by using the obtained knowledge. We can say that BI provides a proactive character to corporations instead of reactive one.

In opposite, an organization without BI can go on, its operational processes can be executed, seems that goals are reach, can even grow up… but when something goes wrong all processes get uncontrolled, it coordination disappears and finally collapses on itself.

We've seen also that even there are some interesting BI solutions in the market, we questioned ourselves about the possibility of develop an own application to perform that function. So, proceeding in that way, instead of parameterize some existing alternative we would have exactly what a company need, so paying more attention to its concrete needs.

As Google offers to developers lots of tools and libraries to use in combination with Google Apps in order to develop new applications, we've seen it as an interesting platform for it. Moreover, as it offers a cloud-computing infrastructure, this fact implies also a reduction on IT costs.

After classify and study the offered alternatives to use in order to create BI-systems, we saw that is possible to easily develop BI systems characterized by its sturdiness, fastness and flexibility. Moreover, these tools will be easy to maintain and update when traffic and data storage needs increase. Due to its architecture, those applications will be focused on the displaying of information in order to know the situation of the company, although the flexibility and change adaptability that them have will allow us to add analysis, forecast and some other tools in case the concrete corporation needs it.

We have also seen that the fact of working just with Google Code tools on the data layer may cause uncertainness to some enterprises because of having its data on

the cloud, and also the scope of solutions can be limited by a strict data store system.

Once again, due to its change adaptability, we saw that if instead of working with data on the cloud, we re-define a data layer composed by various local sources, an ETL process of them to a datawarehouse and the using of some datamarts we would obtain a powerful solution able to process lots of data, avoiding some queries that could collapse the system.

So, all that characteristics, make solutions developed with the described guidelines easily improvable thanks to its independency between data and its visualization. Thus, can be also seen a starting point to develop better ones.

After developing the example tool we verified that is possible to develop a totally functional BI-system for a corporation in a fast and easy way, maintaining all the principles that a software tool should have.

Finally, for ending the conclusions about this project, we can affirm that Google Apps, in combination with Google Code APIs, is an excellent platform for innovate and develop a BI solution for a company totally adapted to its needs, reducing costs, and giving a new perspective over themselves.

Annex: easyBIew code

index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>easyBIew | makes easy what is not</title>
<link href="stylesheets/main.css" rel="stylesheet" type="text/css" />
</head>

<body bgcolor="#666666">
<div id="quad">
        <div id="logform" align="center">
                <img src="easybiewlogo.jpg" />   </div>
        <p id="acces">
                <a href="easybiew">
                Start!
                </a>
        </p>
        <p id="credits">
                <a href="about.html">About easyBIew</a> | <a href="contact.html">Contact
                us</a>
        <br>2010</p>
</div>

</body>
</html>
```

about.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>easyBIew | About easyBIew</title>
<link href="stylesheets/main.css" rel="stylesheet" type="text/css" />
</head>

<body bgcolor="666666">
<div id="quad">
        <div id="logform" align="left">
                <img src="easybiewlogosmall.jpg" />
        </div>
        <p id="seccio">About easyBIew</p>
        <p id="seccio_content">
        <table>
        <tr>
                <td style="width:400" align="justify">
                easyBIew is an example web application, developed under the context of a Diploma
                work, with the aim of
                test some of the alternatives offered by Google<br>in order to build BI-
                systems.<br><br>
                Omar Ghadban Pou<br>
                Univerza v Mariboru, 2010
                </td>
        </tr>
        </table>
<!-- Continue on the following page -->
```

```
        </p>
        <p id="credits">
                <a href="contact.html">Contact us</a> | <a href="easybiew">Go to the menu</a>
        <br>2010</p>
</div>
</body>
</html>
```

contact.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>easyBIew | Contact us</title>
<link href="stylesheets/main.css" rel="stylesheet" type="text/css" />
</head>

<body bgcolor="666666">
<div id="quad">
        <div id="logform" align="left">
                <img src="easybiewlogosmall.jpg" />        </div>
        <p id="seccio">Contact us</p>
        <p id="seccio_content">
        <form action="mailto:easybiew@gmail.com" method="post">
        <table>
        <tr>
                <td>
                Name:
                </td>
                <td>
                <input type="text" name="Name"></input>
                </td>
        </tr>
        <tr>
                <td>
                Surname:
                </td>
                <td>
                <input type="text" name="Surame"></input>
                </td>
        </tr>
        <tr>
                <td>
                Company:
                </td>
                <td>
                <input type="text" name="Company"></input>
                </td>
        </tr>
        <tr>
                <td align="justify">
                Comments:
                </td>
                <td>
                <TEXTAREA rows="4" name="Comments">Type here</TEXTAREA>
                </td>
        </tr>
        </table>
<!-- Continue on the following page -->
```

```
<input type="submit" value="Send">
        </form>
        </p>
        <p id="credits">
                <a href="about.html">About easyBIew</a> | <a href="easybiew">Go to the
                menu</a>
        <br>2010</p>
</div>
</body>
</html>
```

EasybiewServlet.java

```
package easybiew;

import java.io.*;
import java.io.IOException;
import com.google.appengine.api.users.User;
import com.google.appengine.api.users.UserService;
import com.google.appengine.api.users.UserServiceFactory;

import javax.servlet.http.*;

@SuppressWarnings("serial")
public class EasybiewServlet extends HttpServlet {
      public void doGet(HttpServletRequest req, HttpServletResponse resp)
                        throws IOException {

      UserService userService = UserServiceFactory.getUserService();
      resp.setContentType("text/html");
      PrintWriter out=resp.getWriter();
      String thisURL = req.getRequestURI();

      out.println("<html>");
      out.println("<head>");
      out.println("<meta http-equiv=\"Content-Type\" content=\"text/html; charset=iso-8859-1\" />");
      out.println("<title>easyBIew | Login</title>");
      out.println("<link href=\"stylesheets/main.css\" rel=\"stylesheet\" type=\"text/css\" />");
      out.println("</head>");
      out.println("<body bgcolor=\"666666\">");
      out.println("<div id=\"quad\">");
      out.println("<div id=\"logform\" align=\"left\">");
      out.println("<img src=\"easybiewlogosmall.jpg\" />     </div>");
      out.println("<p id=\"acces\">");

      if (req.getUserPrincipal() != null) {

          out.println("<p=\"acces2\">Hello, " +
                        req.getUserPrincipal().getName() +
                        "!  You can <a href=\"" +
                        userService.createLogoutURL(thisURL) +
                        "\">sign out</a>.</p>");

          out.println("</p>");
          out.println("<p id=\"acces\">What do you want to BIew?</p>");
          out.println("<p id=\"acces\"><a href=\"dashboard\">Dashboard</a> | <a
          href=\"reports\">Query&Reporting</a></p>");

      }
```

//Continue on the following page…

```
    else {

        out.println("<p=\"acces\">Please <a href=\"" +
                            userService.createLoginURL(thisURL) +
                            "\">sign in</a> using your Google account.</p>");
        out.println("</p>");
     }


     out.println("<p id=\"credits\"><a href=\"about.html\">About easyBIew</a> | <a
href=\"contact.html\">Contact us</a></p>");
     out.println("<p id=\"credits\">2010</p>");
     out.println("</div>");
     out.println("</body>");
     out.println("</html>");
     out.close();

    }
}
```

dashboardServlet.java

```
package easybiew;

import java.io.*;
import java.io.IOException;
import com.google.appengine.api.users.User;
import com.google.appengine.api.users.UserService;
import com.google.appengine.api.users.UserServiceFactory;

import javax.servlet.http.*;

@SuppressWarnings("serial")
public class dashboardServlet extends HttpServlet {

     public void doGet(HttpServletRequest req, HttpServletResponse resp)
                         throws IOException {

     UserService userService = UserServiceFactory.getUserService();
     resp.setContentType("text/html");
     PrintWriter out=resp.getWriter();
     String thisURL = req.getRequestURI();

     out.println("<html>");
     out.println("<head>");
     out.println("<meta http-equiv=\"Content-Type\" content=\"text/html; charset=iso-8859-1\" />");
     out.println("<title>easyBIew | Dashboard</title>");
     out.println("<link href=\"stylesheets/main.css\" rel=\"stylesheet\" type=\"text/css\" />");
     out.println("<script type=\"text/javascript\" src=\"http://www.google.com/jsapi\"></script>");
     out.println("<script type=\"text/javascript\" >");
     out.println("google.load(\"visualization\", \"1\", {packages:[\"corechart\"]}); ");
     out.println("google.setOnLoadCallback(initialize);");

     out.println("function initialize() {");
     out.println("var query_pie = new
google.visualization.Query('http://spreadsheets.google.com/tq?key=0Ahc6rLjB_skwdDVxdXBqeDVjQWFk
WWIxWThUR3F2QlE&range=A1:B6&gid=0&headers=-1');");
     out.println("var query_line = new
google.visualization.Query('http://spreadsheets.google.com/tq?key=0Ahc6rLjB_skwdDNseTdSVF9uaW9k
VGRiNGdXWUs3UFE&range=A1:C5&gid=0&headers=-1');");
   //Continue on the following page…
```

```
        out.println("query_pie.send(handleQueryPieResponse);");
        out.println("query_line.send(handleQueryLineResponse);");
        out.println("}");

        out.println("function handleQueryPieResponse(response) {");
        out.println("if (response.isError()) { alert('Error in query: ' + response.getMessage() + ' ' +
response.getDetailedMessage()); return;}");
        out.println("var data = response.getDataTable();");
        out.println("var chart = new
google.visualization.PieChart(document.getElementById('chart_div'));");

        out.println("chart.draw(data, {width: 400, height: 300, title: 'Company Expenses'});");
        out.println("}");

        out.println("function handleQueryLineResponse(response) {");
        out.println("if (response.isError()) { alert('Error in query: ' + response.getMessage() + ' ' +
response.getDetailedMessage()); return;}");
        out.println("var data = response.getDataTable();");
        out.println("var chart = new
google.visualization.LineChart(document.getElementById('chart_div_2'));");
        out.println("chart.draw(data, {width: 400, height: 340, title: 'Company Performance'});");
        out.println("}");

        out.println("</script>");
        out.println("</head>");
        out.println("<body bgcolor=\"666666\">");
        out.println("<div id=\"quad\">");
        out.println("<div id=\"logform\" align=\"left\">");
        out.println("<img src=\"easybiewlogosmall.jpg\" />     </div>");
        out.println("<p id=\"acces\">");

        if (req.getUserPrincipal() != null) {

            out.println("<p=\"acces2\">Hello, " +
                                req.getUserPrincipal().getName() +
                                "!  You can <a href=\"" +
                                userService.createLogoutURL(thisURL) +
                                        "\">sign out</a> | go to <a
            href=\"reports\">Query&Reporting</a></p>");

            out.println("</p>");
            out.println("<p id=\"seccio\">Dashboard pannel</p>");
            out.println("<div id=\"chart_div\"></div>");
            out.println("<div id=\"chart_div_2\"></div>");

        } else {

            out.println("<p=\"acces\">Please <a href=\"" +
                                userService.createLoginURL(thisURL) +
                                "\">sign in</a> using your Google account.</p>");
            out.println("</p>");
        }


        out.println("<p id=\"credits\"><a href=\"about.html\">About easyBIew</a> | <a
href=\"contact.html\">Contact us</a><br>2010</p>");
        out.println("</div>");
        out.println("</body>");
        out.println("</html>");
        out.close();
        }
}
```

reportsServlet.java

```
package easybiew;

import java.io.*;
import java.io.IOException;
import com.google.appengine.api.users.User;
import com.google.appengine.api.users.UserService;
import com.google.appengine.api.users.UserServiceFactory;

import javax.servlet.http.*;

@SuppressWarnings("serial")
public class reportsServlet extends HttpServlet {

      public void doGet(HttpServletRequest req, HttpServletResponse resp)
                              throws IOException {

       UserService userService = UserServiceFactory.getUserService();
       resp.setContentType("text/html");
       PrintWriter out=resp.getWriter();
       String thisURL = req.getRequestURI();

       out.println("<html>");
       out.println("<head>");
       out.println("<meta http-equiv=\"Content-Type\" content=\"text/html; charset=iso-8859-1\" />");
       out.println("<title>easyBIew | Query&Reporting</title>");
       out.println("<link href=\"stylesheets/main.css\" rel=\"stylesheet\" type=\"text/css\" />");
       out.println("<script type=\"text/javascript\" src=\"http://www.google.com/jsapi\"></script>");
       out.println("</head>");
       out.println("<body bgcolor=\"666666\">");
       out.println("<div id=\"quad\">");
       out.println("<div id=\"logform\" align=\"left\">");
       out.println("<img src=\"easybiewlogosmall.jpg\" />     </div>");
       out.println("<p id=\"acces\">");

       if (req.getUserPrincipal() != null) {

          out.println("<p=\"acces2\">Hello, " +
                                  req.getUserPrincipal().getName() +
                                  "!  You can <a href=\"" +
                                  userService.createLogoutURL(thisURL) +
                                          "\">sign out</a> | go to <a href=\"dashboard\">Dashboard
          pannel</a></p>");
          out.println("</p>");
          out.println("<p id=\"seccio\">Query&Reporting</p>");
          out.println("<div id=\"form_div\">");
          out.println("<form action=\"reports\" method=\"post\"");
          out.println("<p id=\"seccio_content\">Select the topic:</p>");
          out.println("<select name=\"Topic\">");
          out.println("<option value=\"Sales\">Sales</option>");
          out.println("<option value=\"Expenses\">Expenses</option>");
          out.println("<option value=\"Projects\">Projects</option>");
          out.println("</select>");
          out.println("<p id=\"seccio_content\">Filter by year:</p>");
          out.println("<input type=\"text\" name=\"Year\"><br>");
          out.println("<p id=\"seccio_content\">Filter by month:</p>");
          out.println("<input type=\"text\" name=\"Month\"><br>");
          out.println("<input type=\"submit\" value=\"BIew!\">");
          out.println("</form>");
          out.println("</div>");
//Continue on the following page…
```

```
        } else {

            out.println("<p=\"acces\">Please <a href=\"" +
                                userService.createLoginURL(thisURL) +
                                "\">sign in</a> using your Google account.</p>");
            out.println("</p>");
        }


        out.println("<p id=\"credits\"><a href=\"about.html\">About easyBIew</a> | <a
href=\"contact.html\">Contact us</a>");
        out.println("<br>2010</p>");
        out.println("</div>");
        out.println("</body>");
        out.println("</html>");
        out.close();

    }
    public void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws IOException {

        UserService userService = UserServiceFactory.getUserService();
        resp.setContentType("text/html");
        PrintWriter out=resp.getWriter();
        String thisURL = req.getRequestURI();

        String year= (String)req.getParameter("Year");
        String topic= (String)req.getParameter("Topic");
        String month= (String)req.getParameter("Month");

        out.println("<html>");
        out.println("<head>");
        out.println("<meta http-equiv=\"Content-Type\" content=\"text/html; charset=iso-8859-1\"
/>");
        out.println("<title>easyBIew | Query&Reporting</title>");
        out.println("<link href=\"stylesheets/main.css\" rel=\"stylesheet\" type=\"text/css\" />");
        out.println("<script type=\"text/javascript\" src=\"http://www.google.com/jsapi\"></script>");
        out.println("<script type=\"text/javascript\" >");
        out.println("google.load(\"visualization\", \"1\", {packages:[\"table\"]}); ");
        out.println("google.setOnLoadCallback(initialize);");
        out.println("function initialize() {");

        if(topic.equals("Sales")||topic.equals("Expenses")){
            out.println("var query = new
google.visualization.Query('http://spreadsheets.google.com/tq?key=0Ahc6rLjB_skwdHJaRTB2MG50RTRl
VEFYTzdyTHJlS1E&range=A1:C42&headers=-1&sheet=" + topic + "');");

            if(!year.equals("")){

                if(!month.equals("")){
                out.println("query.setQuery('select B,C where year(A)=" + year + " and month(A)=" +
month + "');");
                }
                else{
                out.println("query.setQuery('select month(A),B,C where year(A)=" + year + "');");
                }
             }
            else{
                if(!month.equals("")){
                out.println("query.setQuery('select year(A),B,C where month(A)=" + month + "');");
                }
//Continue on the following page…
```

```
                else{
                out.println("query.setQuery('select year(A),month(A),B,C');");
                }
            }
        }
        else if(topic.equals("Projects")){
            out.println("var query = new
google.visualization.Query('http://spreadsheets.google.com/tq?key=0Ahc6rLjB_skwdHJaRTB2MG50RTRl
VEFYTzdyTHJlS1E&range=A1:F6&headers=-1&sheet=" + topic + "');");
            if(!year.equals("")){
                if(!month.equals("")){
                    out.println("query.setQuery('select A,B,F where year(E)=" + year + " and
month(E)=" + month + "');");
                }
                else{
                    out.println("query.setQuery('select A,B,F where year(E)=" + year + "');");
                }
            }
            else{
                if(!month.equals("")){
                    out.println("query.setQuery('select A,B,F where month(E)=" + month + "');");
                }
            }
        }
        else{
            out.println("var query = new
google.visualization.Query('http://spreadsheets.google.com/tq?key=0Ahc6rLjB_skwdHJaRTB2MG50RTRl
VEFYTzdyTHJlS1E&range=A1:C42&headers=-1&sheet=" + topic + "');");
        }


        out.println("query.send(handleQueryResponse);");
        out.println("}");
        out.println("function handleQueryResponse(response) {");
        out.println("if (response.isError()) { alert('Error in query: ' + response.getMessage() + ' ' +
response.getDetailedMessage()); return;}");
        out.println("var data = response.getDataTable();");
        out.println("var chart = new google.visualization.Table(document.getElementById('chart_div'));");

        if(!topic.equals("Projects")){
                out.println("var formatter = new google.visualization.BarFormat({width: 80});");

                String col_form;
                if(!year.equals("") && !month.equals("")){
                    col_form="0";
                }
                else if((!year.equals("") && month.equals(""))||(year.equals("") && !month.equals(""))){
                    col_form="1";
                }
                else{
                    col_form="2";
                }

                out.println("formatter.format(data," + col_form + ");");
        }

        out.println("chart.draw(data, {allowHtml: true});");
        out.println("}");
        out.println("</script>");
        out.println("</head>");
```

//Continue on the following page…

```
        out.println("<body bgcolor=\"666666\">");
        out.println("<div id=\"quad\">");
        out.println("<div id=\"logform\" align=\"left\">");
        out.println("<img src=\"easybiewlogosmall.jpg\" />      </div>");
        out.println("<p id=\"acces\">");

        if (req.getUserPrincipal() != null) {

            out.println("<p=\"acces2\">Hello, " +
                                req.getUserPrincipal().getName() +
                                "!  You can <a href=\"" +
                                userService.createLogoutURL(thisURL) +
                                "\">sign out</a> | go to <a href=\"dashboard\">Dashboard
pannel</a></p>");

            out.println("</p>");
            out.println("<p id=\"seccio\">Query&Reporting</p>");
            out.println("<table>");
            out.println("<tr>");
            out.println("<td>");
            out.println("<div id=\"form_div\">");
            out.println("<form action=\"reports\" method=\"post\"");
            out.println("<p id=\"seccio_content\">Select the topic:</p>");
            out.println("<select name=\"Topic\">");
            out.println("<option value=\"Sales\">Sales</option>");
            out.println("<option value=\"Expenses\">Expenses</option>");
            out.println("<option value=\"Projects\">Projects</option>");
            out.println("</select>");
            out.println("<p id=\"seccio_content\">Filter by year:</p>");
            out.println("<input type=\"text\" name=\"Year\"><br>");
            out.println("<p id=\"seccio_content\">Filter by month:</p>");
            out.println("<input type=\"text\" name=\"Month\"><br>");
            out.println("<input type=\"submit\" value=\"BIew!\">");
            out.println("</form>");
            out.println("</div>");
            out.println("</td>");
            out.println("<td>");

            if(topic.equals("Sales")||topic.equals("Expenses")){
                    if(!year.equals("")){
                            if(!month.equals("")){
                                    out.println("<p id=\"seccio_content\">Displaying " + topic +
"&nbspof " + month + " , " + year + " (amount expressed in Milion euro)</p>");
                            }
                            else{
                                    out.println("<p id=\"seccio_content\">Displaying " + topic +
"&nbspof the year " + year + " (amount expressed in Milion euro)</p>");
                            }
                    }
                    else{
                            out.println("<p id=\"seccio_content\">Displaying the complete balance
of " + topic + "&nbspof the company (amount expressed in Milion euro</p>");
                    }
            }
            else{
                    if(!year.equals("")){
                            if(!month.equals("")){
                                    out.println("<p id=\"seccio_content\">Displaying " + topic +
"&nbspstarted on " + month + " , " + year + "</p>");
                            }
```

//Continue on the following page…

```
                        else{
                                out.println("<p id=\"seccio_content\">Displaying " + topic +
"&nbspstarted on the year " + year + "</p>");
                        }
                }
                else{
                        out.println("<p id=\"seccio_content\">Displaying the complete list of " +
topic + "&nbspof the company</p>");
                }
        }

        out.println("<div id=\"chart_div\" style=\"margin-left:40\"></div>");
        out.println("</td>");
        out.println("</tr>");
        out.println("</table>");

    } else {

        out.println("<p=\"acces\">Please <a href=\"" +
                        userService.createLoginURL(thisURL) +
                        "\">sign in</a> using your Google account.</p>");
        out.println("</p>");
    }


    out.println("<p id=\"credits\"><a href=\"about.html\">About easyBIew</a> | <a
href=\"contact.html\">Contact us</a>");
    out.println("<br>2010</p>");
    out.println("</div>");
    out.println("</body>");
    out.println("</html>");
    out.close();


    }

}
```

web.xml

```
<?xml version="1.0" encoding="utf-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">
        <servlet>
                <servlet-name>Easybiew</servlet-name>
                <servlet-class>easybiew.EasybiewServlet</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>Easybiew</servlet-name>
                <url-pattern>/easybiew</url-pattern>
        </servlet-mapping>
        <servlet>
                <servlet-name>dashboard</servlet-name>
                <servlet-class>easybiew.dashboardServlet</servlet-class>
        </servlet>
```

<!-- Continue on the following page -->

```
        <servlet-mapping>
                <servlet-name>dashboard</servlet-name>
                <url-pattern>/dashboard</url-pattern>
        </servlet-mapping>
        <servlet>
                <servlet-name>reports</servlet-name>
                <servlet-class>easybiew.reportsServlet</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>reports</servlet-name>
                <url-pattern>/reports</url-pattern>
        </servlet-mapping>

        <welcome-file-list>
                <welcome-file>index.html</welcome-file>
        </welcome-file-list>
</web-app>
```

main.css

```
#logform {
        margin: 40px;
}
#credits {
        font-size: 10px;
        text-align: center;
}
#acces {
        font-size: 12px;
        text-align: center;
}
#acces2 {
        font-size: 12px;
        text-align: right;
        margin-right: 40px;
}
#seccio {
        font-size: 16px;
        text-align: left;
        margin-left: 40px;
}
#seccio_content {
        font-size: 12px;
        text-align: left;
        margin-left: 40px;
}
#quad {
        background-color: #FFFFFF;
        margin: 80px;
        border: thin solid #000000;
        font-family: Verdana, Arial, Helvetica, sans-serif;
}
```

Acknowledgments

I would like to thank the *Facultat d'Informàtica de Barcelona* and *Univerza v Mariboru* the opportunity they offered me of work abroad, being a great experience in a personal level.

Also, this project would not have been possible without the support of my tutor, Izr. Prof. Dr. Vili Podgorelec. I want to thank him for allowing me to work on a really interesting topic.

Moreover, I would like to thank my professors Jorge Fernández and Carles Farré for making me feel interested about the information systems for organizations and web-based systems disciplines, respectively, and my professor Josep Mª Muntané i Rigol for showing me the real meaning and the importance of take a decision.

Furthermore, I would like to thank Marc Garriga i Portola, computer science engineer of the Barcelona's city council e-Administration department, for showing me the important value of accurate reports about indicators in a big corporation.

Last, but by no means least, I would thank my friends Agustí, José, Francisco, Juan Carlos, Andrea, Begoña, Rosa and Marc for its unconditional help, support and encouragement.

Bibliography

Cited bibliography

[1] Luhn, H. P.: "A Business Intelligence System", October 1958. IBM Journal. Consulted on March 2010.

[2] Power, D. J.: "A brief history of Decision Support Systems, v.4.1", March 2007. DSSResources.com [online]. Available at: http://dssresources.com/history/dsshistory.html. Consulted on March 2010.

[3] Pettey, Christy and Stevens, Hollie: "Gartner Reveals Five Business Intelligence Predictions for 2009 and Beyond", January 2009. Gartner group [online]. Available at: http://www.gartner.com/it/page.jsp?id=856714. Consulted on March 2010.

[4] "¿Qué és Business Intelligence?", 2007. Sinnexus [online]. Available at: http://www.sinnexus.com/business_intelligence/. Consulted on March 2010.

[5] "Google". The free Dictionary by Farlex [online]. Available at: http://encyclopedia.thefreedictionary.com/Google%20Inc. Consulted on April 2010.

[6] "Visión general de la compañia". Google [online]. Available at: http://www.google.es/corporate/. Consulted on April 2010.

[7] "Testimonios de clientes". Google Apps [online]. Available at: http://www.google.com/apps/intl/es/business/customers.html. Consulted on April 2010.

[8] "Herramientas para desarrolladores de Google Apps". Google Apps [online]. Available at: http://code.google.com/intl/es-ES/googleapps/. Consulted on April 2010.

[9] "¿Qué és Google App Engine?". Google code [online]. Available at: http://code.google.com/intl/es/appengine/docs/whatisgoogleappengine.html. Consulted on April 2010.

[10] Davenport, T. and Prusak, L.: "Working Knowledge: How Organizations Manage What They Know", 1998, Harvard Business School Press. Consulted on April 2010.

[11] Fernández, Jorge: "Business Intelligence", 2010. Sistemes d'informació per a organitzacions, FIB, UPC. Consulted on April 2010.

[12] "Ventajas de Google Apps", 2010. Google Apps [online]. Available at: http://www.google.com/apps/intl/es/business/details.html. Consulted on April 2010.

[13] Rotaeche, Estibaliz: "Business Intelligence, Medir para gestionar y aportar conocimiento", 2010. Ibermática [online]. Available at: http://www.ibermatica.com/ibermatica/publicaciones/docComercial/BusinessIntelligence.pdf. Consulted on April 2010.

[14] "Business Intelligence", April 2010. Gestió de Sistemes d'Informació a l'Empresa, blogs UAB [online] Available at: http://blogs.uab.cat/ismanagement/2010/04/20/business-intelligence/. Consulted on April 2010.

[15] Farré, Carles: "From requirements to the UX Model", February 2009. Diseny de sistmes bastats en el web, FIB, UPC [online]. Available at: http://www.slideshare.net/carlesft/dsbw-spring-2009-unit-02-unit-4-from-requirements-to-the-ux-model. Consulted on June 2010.

Non-cited bibliography

"Business Intelligence". Wikipedia [online]. Available at: *http://en.wikipedia.org/wiki/Business_intelligence*. Consulted on March of 2010.

"Google". Wikipedia [online]. Available at: http://en.wikipedia.org/wiki/Google. Consulted on April 2010.

Sánchez Montoya, Ricardo: "Inteligencia de negocios (BI)", March 2003. Monografias.com [online] Available at: http://www.monografias.com/trabajos14/bi/bi.shtml. Consulted on April 2010.

Arias, Emilio: "Exito en la implantación de un sistema de Business Intelligence", July 2007. Monografias.com [online]. Available at: http://www.monografias.com/trabajos29/sistema-business-intelligence/sistema-business-intelligence.shtml. Consulted on April 2010.

"Business Intelligence, now more than ever", 2008. SAP Business Objects whitepapers [online]. Available at: *http://www.businessintelligence.com/fwp/Business_Intelligence_Now_More_Than_Ever.pdf*. Consulted on April 2010.

"Defining Business Analytics and Its Impact On Organizational Decision-Making", February 2009. Computerworld research [online]. Available at: *www.businessintelligence.com/fwp/Defining_Business_Analytics.pdf*. Consulted on April of 2010.

Tappscot, Don: *"Business Intelligence: Actionable Insights for Business Decision Makers"*, 2008. SAP Business Objects whitepapers [online]. Available at: *www.businessintelligence.com/fwp/BI_for_Decision_Makers.pdf*. Consulted on April of 2010.

Elliott, Timo: *"Business Intelligence standarization, executive overview",* 2009. SAP Business Objects whitepapers [online]. Available at: *http://www.businessintelligence.com/fwp/Business_Intelligence_Standardization.pdf.* Consulted on April of 2010.

Fernández Álvarez, A. M.: "Java Servlets", October 2008. Universidad de Oviedo [online]. Available at: http://www.di.uniovi.es/~alberto_mfa/dasdi/?download=04.%20Servlets.pdf. Consulted on May 2010.

"Tutorial de posicionamiento y Layout en CSS", July 2005. CristaLab [online]. Available at: http://www.cristalab.com/tutoriales/tutorial-de-posicionamiento-y-layout-en-css-c111l/. Consulted on May 2010.

Hirst, Tony: "Using Google Spreadsheets as a Database with the Google Visualization API Query Language", May 2009. OUseful.Info, the blog [online]. Available at: http://blog.ouseful.info/2009/05/18/using-google-spreadsheets-as-a-databace-with-the-google-visualisation-api-query-language/. Consulted on June 2010.

Ferg, Steve: *"Java & Python: side by side comparison"*, March 2009. Python conquers the universe [online]. Available at: *http://pythonconquerstheuniverse.wordpress.com/category/java-and-python/*. Consulted on June of 2010.

"Advantages of XML". IBM iSeries information center. Available at: http://publib.boulder.ibm.com/iseries/v5r2/ic2924/index.htm?info/rzakl/rzaklintroadvantages.htm. Consulted on June 2010.

Sol, Selena: "Introduction to XML for web developers". eXtropia [online]. Available at: http://www.extropia.com/tutorials/xml/index.html. Consulted on June 2010.

Google code APIs bibliography

"Google Chart Tools". Google code [online]. Available at: http://code.google.com/intl/es-ES/apis/charttools/index.html. Consulted on April 2010.

"Google App Engine". Google code [online]. Available at: http://code.google.com/intl/es/appengine/. Consulted on April 2010.

"Google Base Data API". Google code [online]. Available at: http://code.google.com/intl/en/apis/base/. Consulted on May 2010.

"Google Secure Data Connector". Google code [online]. Available at: http://code.google.com/intl/es-ES/securedataconnector/. Consulted on May 2010.

"Google Web Toolkit". Google code [online]. Available at: http://code.google.com/intl/es-ES/webtoolkit/. Consulted on May 2010.